C-RWD: A Computational Responsive Web Design Service

Yu Zhang

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 15.11.2020

Supervisor

Prof. Antti Oulasvirta

Advisor

D.Sc. (Tech.) Markku Laine

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Yu Zhang

Title C-RWD: A Computational Responsive Web Design Service

Degree programme ICT Innovation

Major Human-Computer Interaction and Design **Code of major** SCI3020

Supervisor Prof. Antti Oulasvirta

Advisor D.Sc. (Tech.) Markku Laine

Date 15.11.2020 **Number of pages** 102 **Language** English

Abstract

With the increasing diversity of device screen sizes used by web users, responsive web design (RWD) has become an essential part of website production. However, due to its complexity, RWD often requires the cooperation of a team of interface designers, developers and data analysts and requires the team to invest much time. Besides, it is unrealistic to create personalized, responsive designs for users using manual RWD methods.

This thesis proposes C-RWD, which is a computational responsive web design service based on the LaaS platform. The design and implementation of C-RWD are introduced in detail, and the functions of C-RWD are evaluated based on example websites. When an initial interface is provided, C-RWD can (1) automatically collect the interaction data of different users in the interface and generate personalized breakpoints and (2) automatically use combinatorial optimization to generate optimized interfaces at these breakpoints based on user's interaction data to improve usability and then (3) automatically integrate the optimized interfaces under different breakpoint widths into one fully responsive website and present it to users.

The contribution of C-RWD is that it proposes a novel framework which can automate the generation of responsive interfaces while optimizing the interface for user's personalization. In practical applications, it can reduce the time and cost of RWD production and generate a personalized and optimized interface for users.

Keywords Responsive Web Design, Layout Personalization, Combinatorial Optimization, Automated Interface Generation

Preface

I would like to first thank my supervisor Prof. Antti Oulasvirta for giving me the opportunity to join the user interfaces research group to conduct such an interesting and challenging thesis topic. He pointed out the research direction and provided professional opinions on it. Then, I would like to express my special thanks to my advisor, D.Sc. Markku Laine. His guidance and strong support helped me complete this thesis. Besides, I would also like to thank D.Sc. Ai Nakajima, M.Sc. Simo Santala and other colleagues in the research group who always enthusiastically and patiently shared their knowledge and insights when I encountered difficulties and asked for help.

Finally, I would like to express my gratitude to my family and friends. Their support and encouragement help me overcome difficulties during the thesis process. I want to thank my girlfriend, Xiaoyun Zhang, for her accompany and love throughout my thesis working process.

Espoo, 15.11.2020

Yu Zhang

Contents

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
BFC	Blocking Formatting Context
C-RWD	Computational Responsive Web Design
CPL	Characters Per Line
CSS	Cascading Style Sheets
DIP	Device-Independent Pixel
DOM	Document Object Model
DPI	Dots Per Inch
DTG	Design Task Generator
E-commerce	Electronic Commerce
GUI	Graphical User Interface
HTML	HyperText Markup Language
ILP	Integer Linear Programming
IP	Integer Programming
JS	JavaScript
JSON	JavaScript Object Notation
LaaS	Layout as a Service
NLP	Natural Language Processing
PC	Personal Computer
PPI	Pixels Per Inch
RQ	Research Question
RWD	Responsive Web Design
SQL	Structured Query Language
UI	User Interface
W3C	World Wide Web Consortium
WCAG	Web Content Accessibility Guidelines

1 Introduction

In this section, the research motivation, research objectives, research questions (RQ), and research approach of Computational Responsive Web Design (C-RWD) will be introduced. Then, the structure of this thesis will be briefly explained to facilitate reading.

1.1 Motivation

With the increase of mobile data traffic and the emergence of display devices of different widths [1, 2], responsive web design (RWD) has become a necessary design consideration for website design. However, designing and developing responsive websites often dramatically increases the difficulty of website production. Designers and developers need to iteratively design and develop interfaces of different sizes to achieve an excellent responsive experience in close collaboration. Despite the help of various responsive frameworks, this process is often tedious and time-consuming. Due to the increase in difficulty, responsive web design is mostly unfriendly to the web design and development team to maintain good website usability. Therefore, for stakeholders to create a new responsive website from scratch or already have a non-responsive website, a new service is needed to generate a responsive layout from a static layout automatically. For stakeholders who have created a responsive website, a new service is demanded to help them continuously optimize the website's responsive performance.

Computational methods, such as integer programming, is a method that can explore the design space in a limited time and can guarantee the optimal solution. Computational methods have been used to optimize the design of the graphic user interface [3], layout restructure [4] and interactive layout design support [5]. Recently, Laine et al. proposed the Layout as a Service (LaaS) platform, which provides web page layout self-optimizing services through computational methods [6]. This platform uses Fitts' law and other models to set the objective function to optimize the interface's layout and regenerate the optimized layout into a static interface, which greatly improves the usability for the individual user. Due to this platform's scalability, computational responsive web design can be added as a service to realize the generation and optimization of responsive interfaces.

In summary, designers and developers of responsive websites have demands for services that help in RWD, and computational methods have performed well in previous

GUI optimization studies. So, this thesis proposes Computational Responsive Web Design (C-RWD): a responsive web design service using computational methods. This service can help users create and optimize responsive websites. It can help users save the time and cost required for RWD and optimize the interface layout. This service can be accessed through the LaaS [6] platform.

1.2 Research Objective

This thesis aims to design and develop C-RWD, which can be used as a service of LaaS to automatically generate and optimize fully responsive layouts for an input web page. The high flexibility of the responsive user interface brings challenges to the automatic optimization and generation of the interface. According to related literature search, there are some research or techniques trying to automate the RWD process or optimize interface layout. However, no related work can automate the responsive interface generation while optimizing the interface layout. This research will focus on the automated generation and layout optimization processes of RWD. C-RWD is positioned as a prototype service that uses computational methods to achieve RWD, rather than a product-level service. So, this research will not involve topics such as responsive interface loading performance, cross-browser compatibility, and web security.

1.3 Research Questions

From the research motivation and research objective, the research questions(RQ) are listed as below:

- **RQ1: Given one input interface, how to automatically generate responsive breakpoints for different users?**
- **RQ2: Given one input interface, how to automatically generate optimized layouts in different breakpoints width to improve the usability?**
- **RQ3: Given one input interface, responsive breakpoints from RQ1 and optimized layouts from RQ2, how to automatically adapt the interface to be responsive?**

For RQ1, C-RWD must be able to generate customized breakpoints for different users. These breakpoints will be used as the responsive breakpoints of the final

output responsive interface. For RQ2, C-RWD must be able to optimize the input interface at different width. In this way, C-RWD could provide optimized layouts with improved usability for the input interface at each breakpoint. For RQ3, when the input interface provided by the user is static or semi-static, C-RWD needs to perform proper responsive adaptation on this interface using the optimized layouts from RQ2 to generate a responsive interface with breakpoints generated in RQ1. When the input interface is already responsive, C-RWD needs to redesign the responsive interface with the optimized layouts from RQ2 and breakpoints generated from RQ1.

By answering these research questions, the final research question which is listed as below should be solved:

- **Final RQ: Given one input interface, how to automate the generation of an optimized responsive layout with improved usability?**

1.4 Research Approach

This thesis uses design science as the research approach [7]. It firstly researches and analyzes RWD related theories and development techniques. Based on the understanding and comparison of different RWD design theories and implementation technologies, the C-RWD system is designed and implemented to improve the RWD process and result. Finally, this thesis will evaluate the C-RWD system and generated results to verify whether the research question is well answered.

1.5 Structure of Thesis

Section 1 of this thesis briefly introduces the research motivation, research questions, and research approach. Section 2 mainly introduces the theoretical background of C-RWD. It first introduces the underlying web technologies. Then, from the historical perspective of web layout design and development, it introduces various web layout methods in chronological order. Finally, it introduces the application of computational methods in user interface design. Section 3 focuses on the RWD concepts and analyzes the key elements, frameworks, and services in the RWD process. Section 4 goes deep into the design and implementation of the C-RWD system and elaborates on the key ideas and technical details. Section 5 lists and evaluates the interfaces generated by C-RWD. Section 6 discusses and analyzes related research and technologies and compares them with C-RWD. Section 7 summarizes the whole thesis and reviews the research questions again.

2 Background

This section introduces the background knowledge of C-RWD, including the development and history of web design, commonly used web development techniques, and some applications of computational methods in web design.

2.1 Cornerstone Technologies of Web Development

This subsection mainly introduces commonly used web development technologies: HyperText Markup Language(HTML), Cascading Style Sheets(CSS), JavaScript. These technologies are the foundation of responsive web development. Many of the features of these technologies are applied in the process of C-RWD's responsive conversion.

2.1.1 HyperText Markup Language

Hypertext markup language, known as HTML, is a markup language rather than a programming language. It can not only express text information but also store and express hyperlink information. Therefore it is called hypertext markup language. It is usually used for web development. HTML has many different types of elements such as `div`, `p`, `image`, etc. The name of the element is surrounded by angle brackets to become an element tag. Each HTML element usually consists of a start tag, end tag, text content, and element attributes. Developers can use different element combinations and nesting to build pages according to their needs. The web browser can read the HTML file information and render the HTML elements to the page.

HTML has went through multiple versions of evolution, and the latest version is now HTML5. HTML5 brings many new features and APIs such as semantic elements, canvas elements, new multimedia elements such as `<video>` and `<audio>`, detection of device orientation [8].

HTML builds the essential part of web page. However, using HTML alone often fails to meet modern web development's advanced needs, such as dynamic pages and sophisticated styles. Therefore, it is often used in conjunction with JavaScript, CSS, and other technologies to build complete web pages [9]. HTML is usually used as the skeleton of the page to form the main body of the page. Its elements style is defined by the CSS file, and the behavior of the elements is defined by the JavaScript file. These two techniques will be introduced below.

2.1.2 Cascading Style Sheets

CSS is short for Cascading Style Sheet. In the development of web pages, HTML is used to express the structure and content of the web page, while CSS is used to design and express the page's style. CSS cannot be used alone, and it must be combined with HTML to build a complete web page. Figure 1 shows an example of CSS code with corresponding HTML. To clarify the relationship between CSS and HTML, Figure 2 shows the difference between enabling CSS styles and canceling CSS styles for the same HTML code in Figure 1.

HTML:

```
<div class="card">
    <h1>
        Hello, world!
    </h1>
    <p>
        Nice to meet you!
    </p>
</div>
```

CSS:

```
.card{
    background: #FFCD00;
    padding: 10px;
    width: 200px;
    border: 5px solid #005EB8;
}
.card h1{
    color: #EF353B;
}
.card p{
    font-size: large;
    color: white;
}
```

Figure 1: A piece of CSS code example with HTML code.

In the early days of HTML, web developers have begun to use various style sheets to modify web pages, but there is no unified standard to regulate such style sheets. CSS was first proposed by Håkon Wium Lie in 1994 [10]. In 1996, the first version of the CSS standard CSS 1 was published as a W3C recommended standard [11]. After that, the CSS standard went through several iterations and updates. The current stable version is CSS 2.1, and CSS 3 has been accepted by most browsers and widely used by developers. For browsers that do not fully support the CSS 3 version, developers can add different browser prefixes to let them use the older version's CSS

Figure 2: Comparison between disabling and enabling CSS style for same HTML: (a) HTML code without CSS; (b) HTML code with CSS.

features.

Compared with other style sheets, CSS has become popular due to its cascading feature and has become an essential web development technology. The word "cascading" means that multiple style sheets and inline styles can control a web page's style. The cascading styles from different sources modify the final web page style according to the order of execution, the combination of different selectors with different priorities, and the `!important` keyword according to different priorities. In this way, web page developers can add or adjust web page styles flexibly and conveniently, and web page users can also adjust web page styles in a cascading manner according to their needs. Usually, only one code line is changed to replace the style sheet being used by the web page. Besides that, CSS also has many other advantages.

For example, CSS has achieved the separation of web content and styles. Compared with using inline styles in HTML, CSS is easier to manage global styles and can ensure consistent styles. For example, using CSS can easily set the default global font style of the page. This feature facilitates the iterative development of web pages and helps improve the usability of web pages. However, when the page's style is very sophisticated, the CSS will become quite verbose, and there will be many repeated attributes and code blocks, which also leads to the increase in the difficulty of CSS code management and potential style conflicts. Some CSS preprocessing tools such as

SASS [1] and LESS [2] have been invented to solve this problem through features such as `mixin` and style nesting supporting. These tools could help developers organize a CSS file's structure and write intricate styles elegantly. Because these tools will parse the code into standard CSS code, there are no browser support issues when using these state of art technologies.

2.1.3 JavaScript

JavaScript, JS for short, is a light weight interpreted language mainly used for browser script [12]. It was created by Brendan Eich in 1995 in about ten days [13]. After that, JavaScript was formally proposed by Netscape and Sun [14].

JavaScript was initially designed to solve some simple form verification problems on the client-side. Because these problems can only be solved on the server-side, web users need to wait for the network transmission time after submitting form verification. During that period, the internet bandwidth was often low, which resulted in poor user experiences such as long and frequent waiting times. Therefore, a browser-side scripting language that mainly runs on the browser side is required, which is why JavaScript was created.

Now, in addition to form validation, JavaScript is also mainly used to control the interactive behavior of the page, such as listening to page events and reacting to page events and dynamically changing page content or structure. With the JavaScript version change, JavaScript has become more and more powerful and has more functions. JavaScript can even be used in server-side development. For example, Node.js allows JavaScript to run without the browser. The using scenarios of JavaScript has also been significantly increased.

In the early stages of its birth, JavaScript has multiple versions of implementation. Therefore, in 1997, the Ecma General Assembly proposed a specification for the JavaScript language and named this specification ECMA-262. The programming language specified by the ECMA-262 standard is called ECMAScript. JavaScript is one of the most successful implementations of ECMAScript. Currently, the latest ECMAScript version is ECMAScript 2020 [3]. However, there are many latest features which are not supported by browsers. There are some tools like Babel [4] which can convert the latest JavaScript version code into browser compatible old version. So,

[1] https://sass-lang.com/
[2] http://lesscss.org/
[3] http://www.ecma-international.org/ecma-262/11.0/index.html
[4] https://babeljs.io/

developers could use latest features of JavaScript in their projects assisted by these compiling tools.

2.2 A Brief History of Web Design

In this subsection, the evolution history of web layout design is reviewed and introduced. In the history of web design, due to the development of new technologies and features and the evolution of user equipment, the method of web design is also constantly evolving and changing. One of the most significant changes is that with the popularity of mobile devices and the diversified development of user device screen sizes, responsive web design has changed from optional design consideration to necessary design consideration. The fixed-width layout design method began to hurt the user experience. Before officially introducing the concept of C-RWD, it is necessary to review and summarize the development history of the web design. The different layout types mentioned below are mainly based on part of the book [15]. These different interface layout design methods are smoothly introduced by time.

2.2.1 Fixed-Width Layout

The main idea of the fixed-width layout is to make one layout design with fixed width elements. It is possible to make a pixel perfect layout. The fixed-width layout method is very popular in the early web design period because at that time, mobile devices have not yet become popular. The display device size range of the web page is limited, usually having a resolution around 1024x768 and 800x600 according to [16]. Web designers do not need to consider the issue of displaying web pages on mobile devices with narrow screens. Therefore, the designer can make the page body to a certain fixed pixel value, usually around 960 pixels, and then set the margin to make the page centered. The margin trick used in this process is that by setting margin style as `margin: 0 auto`. According to the [17], the browser will automatically calculate the left and right margin values and put the block horizontally center position. When the viewport width is wider than the fixed value, there will be empty white space displayed at the left and right side of the content. When the viewport width is smaller than the fixed width, a horizontal scroll bar will appear to help users browse all the content. Figure 3 shows the structure of a typical fixed-width layout.

According to [15], the benefits of using fixed-width layout are that designer has full control of the layout and it is easy to develop this kind of layout. The designer only needs to design a fixed-width layout. And the developer only needs to develop and

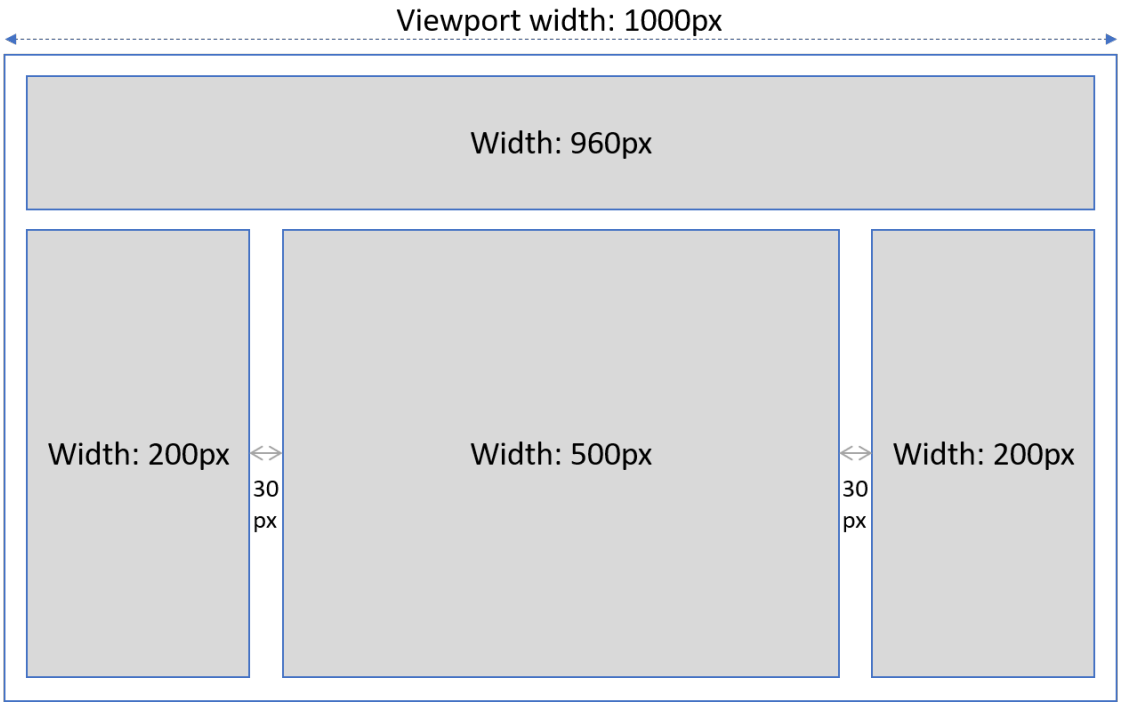


Figure 3: Fixed-width layout structure.

test for the fixed-width layout. However, there are also many drawbacks brought by this kind of layout methods. For example, there will be empty white areas when the layout is displayed on a widescreen. This will lead to a waste of screen space. When displayed on a narrow screen, the horizontal scroll will also make poor user experience and users need to scroll it frequently to browse the content.

2.2.2 Variable Fixed-Width Layout

The main idea of the variable fixed-width layout is to use JavaScript code to dynamic switch multiple pixel-perfect fixed-width layouts. This method predefined different layout and content for the different range of screen resolutions. When the code detects user screen resolution, it will automatically display the corresponding layout for that resolution range.

The advantage of this method is it has better usability than a fixed-width layout because it provides a specific layout design for different screen range. But the disadvantages are it could not cover all the resolution range situation and also it will bring repeated work for web designers and developers. Besides, it has the same advantages and disadvantages as the fixed-width layout has.

2.2.3 Fluid Layout

The idea of the fluid layout is to make the entire layout flow like liquid. In order to achieve this goal, the fluid layout uses percentages as the element's position unit, that is, relative positioning units are used to make the page layout related to the width of the page viewport. In this way, when the user adjusts the width of the viewport, the fluid layout will always fill the entire window to make full use of the space of the viewport. Web developers can use `min-width` and `max-width` to constrain the maximum and minimum widths of elements to further control the display of the page. Figure 4 shows the structure of a fluid layout example.

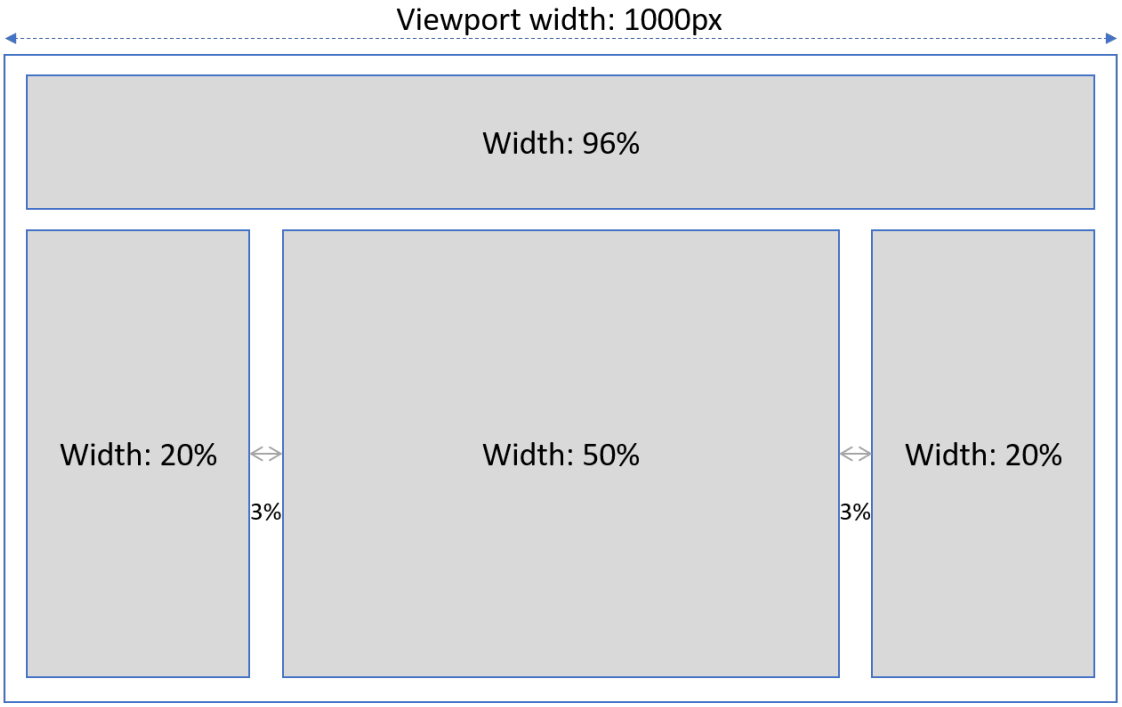


Figure 4: Fluid layout structure.

The advantages of using fluid layout are as follows: First, as mentioned above, the layout can make full use of the viewport space. This can also avoid horizontal scroll bars or blank areas on the left and right sides. This usually improves the usability of the page. Secondly, for users, they can adjust the width of the page themselves to suit the needs of the usage scenario. This can also adjust the amount of text per line to improve the readability of the page text content. Also, compared to the variable fixed-width layout, the fluid layout ensures the consistency of the layout while adjusting the page size. The disadvantages of fluid layout are as follows. First of all, fluid layout requires developers to have a clear and profound understanding of

the structure of the page. Because developers need to understand the relationship between the elements to accurately calculate the relative position of each element. Secondly, when using percentage as the unit, there will be values such as "33.3333%" that need to be rounded with a certain precision. And this round process may cause some unpredictable problems in the rendering of the page, especially on some boundary problems. Finally, the display of the fluid layout is related to the width of the page viewport, which brings benefits and uncertainty. Web designers cannot fully predict the performance of pages in various widths. The usability of web pages may drop sharply in some extreme cases, such as extremely narrow viewport widths.

2.2.4 Fluid Grid Layout

The fluid grid layout was first proposed by Ethan Marcotte in 2009 [18]. The main idea of this layout method is to transform a grid-based fixed-width layout into a fluid grid layout through the fluid layout method. In the fixed-width grid layout, the width of each column of the grid is fixed, that is, the absolute unit pixel is used. But in the fluid grid layout, the grid is first designed with a fixed unit, and then a simple formula: target/context = result from [15], is used to convert the absolute unit to a relative value based on percentage.

The advantage of a fluid grid layout is that when the viewport width changes, it maintains the design of the grid, such as maintaining the position of each element in the grid. It combines the advantages of a fixed-width grid and fluid layout. However, to do this, it requires unit conversion for each element.

2.2.5 Adaptive Fluid Layout

When the width of the page viewport varies widely, the traditional fluid layout often cannot simultaneously take into account the excellent design performance and usability on narrow screens such as mobile devices and wide screens such as PC monitors. Therefore, the use of adaptive fluid layout is to solve this problem. Its basic idea is to design multiple fluid layouts for different viewport width ranges and detect changes in the user's viewport width through JavaScript code to switch between different designs.

The advantage of this layout is that it can improve the overall usability of the fluid layout by switching among multiple designs. The adaptive fluid layout also allows web designers to have more comprehensive control over web pages' performance. However, multiple designs also bring repetitive work, and still cannot cover all screen

sizes. Furthermore, the adaptive fluid layout still has some shortcomings of the fluid layout, such as the need to use JavaScript code to detect page width changes.

2.2.6 Elastic Layout

An elastic layout's basic idea is to use the relative unit `em` to position all elements. The CSS unit `em` is a particular relative unit, and its calculation formula is" em = the size of the target element in pixels/the font size of the parent element in pixels", according to [19]. In this way, when the size of all elements uses em as the unit, the page's size becomes related to the font size of the page instead of the width of the viewport. When the user adjusts the page's font size, the layout of the page will also change to maintain the original proportion relationship of the design. A design trick is to set the font size of the body element to 0.625em so that the default 1em size of its child elements is equivalent to 10 pixels, which can facilitate web designers' calculation.

The main advantage of an elastic layout is that if appropriately developed, the elastic layout allows users to adjust the page's size according to their needs and maintain good page usability. The main disadvantage is the increase in development difficulty and page complexity. When the page can change in real-time with the font size, developers often need a lot of calculation and testing to adjust the element's em value. Development using elastic layout also requires developers to have a good understanding of the structure of the page layout. Besides, when the font size changes to some specific range, the elastic layout may still have horizontal scroll bars or white space on the left and right sides. This kind of screen space issue also affects the user experience a lot.

2.2.7 Hybrid Layout

The hybrid layout's basic idea is to reasonably combine multiple layout methods on different page components to connect the advantages of these layout methods and avoid their disadvantages. This method has higher requirements for web designers and developers because they need to understand and be familiar with the suitable scenarios and their respective advantages and disadvantages of various layout methods and select and combine them correctly. For example, in a hybrid layout, its logo part often needs to be laid out using a fixed-width method because, in many cases, the size of the logo element on the page does not change with the page width. Furthermore, its content section can use the fluid layout or elastic layout to improve

the page's usability in different widths. Its overall layout can use the adaptive fluid layout method to carry out different layout designs for different page width ranges to optimize usability further.

The advantages of hybrid layout are apparent. It can flexibly combine multiple layout methods to make the page have a good layout. Its ideas are still widely used in web layout design today. For example, web developers can use the latest grid layout to design the page's overall layout and use the flexbox layout to create the internal layout of the page components(see below). Its disadvantage is that it has higher requirements for web designers and developers, and requires much design and development experience to use it well.

2.2.8 Flexbox Layout

In general, the flexbox layout is a one-dimensional layout method, although it can achieve a two-dimensional layout effect by nesting multiple elements. The flexbox layout usually consists of two types of elements: flex container and flex item. The flex container is a container element that wraps flexbox items, and the flex item is a direct child element in the flexbox. The flex container controls the flex item through two axes: the main axis and cross axis. By setting the `flex-wrap` property, flex container can set whether these elements will wrap under certain circumstances. The flex items are placed in the main axis from the main start to the main end according to the set order attribute. When the size of the flex container element in the main axis direction changes, the flex item element undergoes expansion and contraction changes through the preset `flex-grow`, `flex-shrink`, and `flex-basis` attributes. The cross axis is responsible for controlling the flex items in the cross direction, for example, by setting `align-content` to control the layout of multiple rows of flex items in the cross axis direction. The flexbox layout structure is shown as Figure 5.

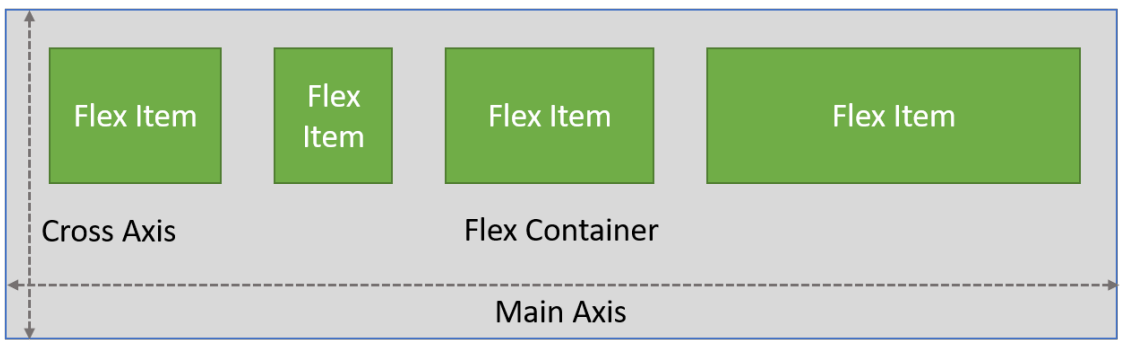


Figure 5: Flexbox layout structure.

The flexbox design concept is a content-first layout method, which means there is content first, and then the content is laid out into a form of content flow according to the rules of the flexbox. It does not focus on the overall content layout and precise positioning but allows the content to be naturally placed in the flex container's content flow.

Flexbox layout has many advantages. First of all, it can help locate and layout smaller-scale components and is especially good at operating one-dimensional elements. For example, for a web page's navigation bar component, the flexbox can easily create a flexible navigation bar and manipulate its elements to be stretched, proportions maintained, and vertically centered. Secondly, flexbox can quickly achieve positioning effects that cannot be achieved using traditional float and position methods, such as vertically centering a child element. Flexbox reduces the development difficulty for developers and complements the traditional CSS positioning skeleton. Flexbox layout also has some weaknesses. For example, because it focuses on the content flow rather than the overall precise layout and outlining, it is not suitable for complicated web layout. Besides, the one-dimensional layout characteristics make it unable to fulfill some two-dimensional positioning requirements.

Flexbox layout belongs to the W3C standard of CSS Flexible Box Layout Module Level 1 [20]. This standard is a W3C candidate recommendation proposed in November 2018. Although the draft of the flexbox layout was proposed in 2009, browsers had poor support for this layout method in the early stages, and so it was not widely used. Nowadays, according to statistics [21] from the "CAN I USE" website, nearly all of the major browsers already support this feature, making it a mature and widely used layout method. The statistics figure is shown as Figure 6.

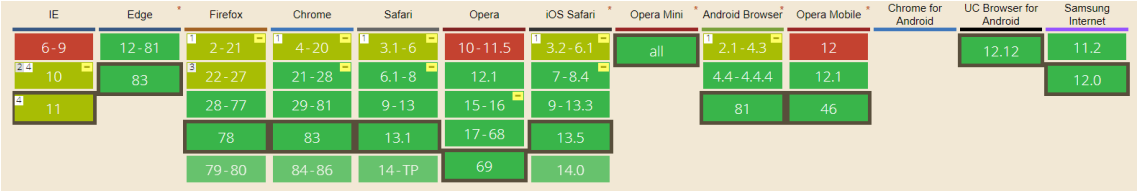


Figure 6: Flexbox browser support statistics from CAN I USE website.

2.2.9 Grid Layout

The grid layout's main design idea is to lay out the entire page and position the elements through the grid. Therefore, it is a two-dimensional layout method that can accurately position elements. The grid layout consists of two types of ele-

ments. The grid container element is used as a container to wrap the grid items element. By setting `display: grid`, an element is set to grid container. The grid template concept is used here to control the design of grid rows and columns and grid area. These designs are all completed in this grid container. By setting the `grid-template-columns`,`grid-template-rows`, and `grid-template-areas` properties of the grid container, users can achieve the design of the grid's rows, columns, and grid area. The grid item is the child element of the grid container, which is the web design's content element. After completing the grid template design, these elements can be accurately positioned and placed in the grid. By setting the `grid-column`, `grid-row` and `grid-area` of the grid item, developers can specify the position of the grid item in the grid. A standard interface using grid layout is shown in Figure 7.

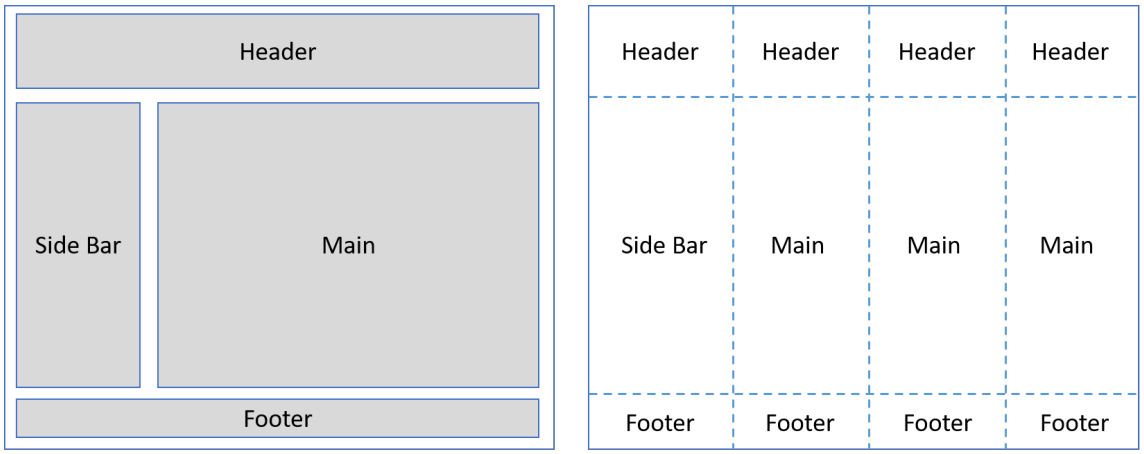


Figure 7: Grid layout structure.

Grid layout uses layout first strategy, which means that designer first designs and builds the whole layout grid framework and then puts the web content into the grid. It is a two-dimensional layout method and can control items both in terms of row number and column numbers. It focuses on the overall layout control and not on a single component. Because it has complementary features with flexbox layout, their two layout methods are usually combined. For small-scale component-level elements, it is better to use flexbox for layout in a web design process. It is more suitable for a large-scale and complex layout to use the grid layout for the overall layout.

Grid layout has many advantages. First, it can accurately organization the entire page from a two-dimensional perspective. Second, it can control the gap between the grids to meet the spacing requirements between elements in the layout. The grid layout's disadvantage is that not all browsers can support all features of the grid layout well. However, in the future, with better browser support, grid layout will be

one of the most popular choices for interface layout.

The grid layout was proposed in 2012 as a working draft of W3C, and now the latest standard version is CSS Grid Layout Module Level 1, which was proposed in 2017 as W3C candidate recommendation [22]. Since the grid layout was proposed later than the flex layout, the browser's support level is slightly worse than the flex layout. Nevertheless, in recent years, most browsers have also supported this new layout. According to statistics from the "CAN I USE" website, most of the latest browser versions already support grid layout. It means that web developers can use grid layout to layout the page in the current daily development. The statistic figure is shown as Figure 8.

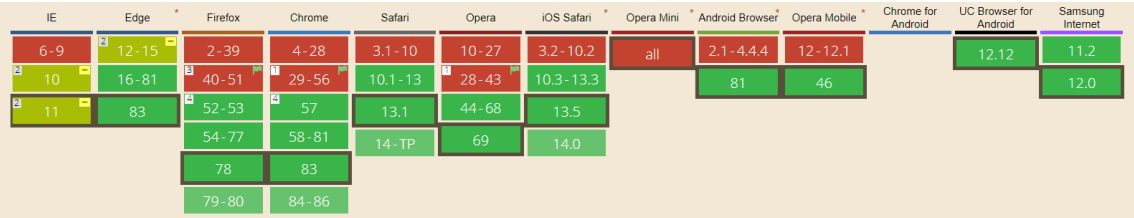


Figure 8: Grid layout browser support statistics from CAN I USE website.

2.2.10 Responsive Layout

Now it is time for responsive layout. The basic idea of a responsive layout is that the web interface should change with the context of the user's device. For example, as the types of user devices are different, the web interface should adjust its layout to adapt to these devices of different widths and resolutions. Besides, the user's context may also include the environment in which the user browses the web interface, such as light intensity and other factors. When the user is in an environment with low light intensity, the responsive interface can sense the light intensity through the sensor and adjust the interface's theme color. The responsive layout does not limit the layout technology used. A web interface usually uses multiple technologies such as grid layout, flexbox, and JavaScript code to achieve rich responsive features.

Since the responsive layout is the foundation of C-RWD, the specific concept and detailed knowledge of the responsive layout will be elaborated in Section 3.

2.3 Computational Methods in User Interface Design

2.3.1 Combinatorial Optimization Overview

Combinatorial optimization is an optimization method to solve optimization problems with discrete variables. Its goal is to find the optimal solution to maximize or minimize the objective function in the finite solution space [23]. The objective function is used to describe what kind of solution is pursued in the optimization process. For each candidate solution in the search space during optimization, the objective function will evaluate it and return an objective value [24]. In the minimization optimization, the candidate with the lowest objective value is the optimal solution. In maximum optimization, the candidate with the highest objective value is the optimal solution. During the optimization process, the design constraint can help decrease search space since the candidate which does not meet the requirement of design constraints will never be a optimal solution [24].

Combinatorial optimization has two mainstream optimization methods [24, 25]. The first is the exact optimization method. When the optimization problem's objective function is given, the exact method can be used to find the optimal solution to the optimization problem. Usually, the exact method is used for P problems which can be solved in polynomial time. A polynomial can express the objective function of this kind of problem. However, the exact method can also solve NP problems and NP-hard problems, although the required solution time may reach an exponential level, which is difficult to use under the existing computing power. The Exact method can guarantee the optimal solution of the optimization problem in a limited time, which is also the most attractive feature of the exact method. There are some common exact methods, such as cutting plane method, branch and bound method.

The second is the black-box optimization method, also called the heuristic method. As it means, black-box optimization does not make any settings for the optimized objective function but treats it as a black box. After putting a candidate optimization candidate in the optimization process into this black box, it can give the optimization candidate the evaluation result and use this to determine whether to update the current global optimal optimization candidate result. Black-box optimization is often used for optimization problems where computational complexity and time increase exponentially as the problem's scale increases. It needs an objective function that can be calculated at low-cost [24], because it needs to continually put solution candidates into the objective function for evaluation to find the approximate optimal solution.

The black-box method cannot guarantee that the final result is the global optimal solution. There is often a trade-off between the search time and the accuracy of the final solution. With more calculation time invested in the black box method, it can usually find the global optimal solution with a higher probability. Conversely, less calculation time often leads to poor search results, and the black-box optimization method may return a locally optimal solution that deviates significantly from the global optimal solution. Some meta-heuristics, such as the simulated annealing algorithm, are proposed to solve the local optimal solution problem of the black-box method, [24].

In short, black-box optimization cannot guarantee to return to the global optimal solution. However, in many problems, a solution close to the global optimal solution is often sufficient. Consequently, black-box optimization also has many application scenarios. In addition to the simulated annealing algorithm, there are other heuristic optimization algorithms, such as hill climbing and greedy algorithms.

2.3.2 Combinatorial Optimization for User Interface Design

The user interface design problem is a typical NP-hard problem. Because there are too many decisions that need to be considered in the user interface design process. There are decisions such as the position, color, size, semantics of the layout elements. There will be exponential situations only considering the placement of variable elements in a fixed-size canvas. Therefore, due to the vast design space, the optimization problem has a vast search space, so that it seems complicated to use the exact method to solve this problem. For heuristic optimization methods, the evaluation of user interface design schemes is often not low-cost, making it challenging to use heuristic optimization methods to explore the approximate optimal design.

However, just like the process of user interface design, the optimal design is by no means achieved overnight. Iterative updates make the design approaching the optimal design regarding many different aspects. It is unrealistic to optimize all aspects of web design and find the optimal solution through one optimization process. However, when the user interface design's optimization problem is decomposed into modular optimization problems of various aspects, the exact method is quite promising and has much work to do. For example, Fitts' law [26] can be utilized to optimize the time and accuracy of target selection in the interface [3]. For another instance, the salience model can control the significance of a specific element in the interface to help users find it faster. Besides, personalized optimization results can be achieved by

customizing these models' specific parameters for individual users, such as adjusting the importance of different interface elements through the user's interactive behavior and historical records.

Although it dramatically reduces the problem's scale by optimizing only part of the design goals for interface design at a time, the time to find the optimal solution is often still unacceptable. As the problem of placing uncertain size elements in a fixed-size interface mentioned above, this optimization problem still has an exponential solution space. However, the unique constraints of interface design can help solve this optimization problem. When designing a user interface, the elements must not overlap each other, be aligned horizontally or vertically, and have some subordination and semantic relationships. These constraints can be applied to the process of interface optimization. According to this survey [27], some typical constraints in page design such as abstract constraint and spatial constraint are utilized further to tighten the design space of interface design optimization problems. At the same time, mathematical methods can be easily used to express these constraints. For example, by setting the relationship between the width and height of the x and y coordinates of the interface elements, the elements will not overlap each other.

Because of the model-based objective function and the specific constraint method, the interface design optimization problem can use the exact method to find a globally optimal solution. In practice design, subtle variations are usually difficult noticed by users and seldom have a substantial impact. Many of the final optimization requirements have a certain degree of relaxation, which further reduces the time to search for the optimal solution. Therefore, through the integer programming or mixed-integer programming in the exact method, some commercial optimization solvers such as Gurobi [28] can obtain good optimization results for user interface design optimization in limited time and computing resources.

According to [3], combinatorial optimization can be used first, to generate a new optimized interface design according to a specific objective function. Second, it can also enable the designer in the loop to interactively assist them in interface design, such as by continually providing designers with candidates for the next design based on the current context. In this process, the designer still takes the lead in the design, and the optimization system serves as a supporting role to improve the quality of the design. Third, as mentioned above, by personalizing the optimization process's parameter, such as the importance of interface elements, the optimization process can produce personalized optimization results for different users. Fourth, combinatorial

optimization can help verify whether the current interface design is optimal or close to the optimal solution based on a specific design goal and the objective function.

2.3.3 Machine Learning for User Interface Design

In addition to combinatorial optimization, machine learning can also help design and develop user interfaces.

In the interface design and prototype stage, machine learning can help designers get more ideas through the interactive design process. For example, it can help designers generate inspiration and design ideation through cooperative contextual bandits. [29]. In the development stage of the user interface, machine learning can automatically recognize and convert the prototype into the web page's code framework through the neural network [30]. After the interface development is completed, machine learning can test the web page's components to improve the quality of the page and find potential errors [31]. Also, machine learning can help combinatorial optimization select appropriate optimization parameters [3] through training on the corresponding data set.

However, the method of using machine learning currently has some limitations. First, it often requires a large number of data sets for training. For user interface design and development, the user interface's high complexity makes the collection and labeling of corresponding data expensive. Secondly, this is a data-driven method, but the trained model does not necessarily guarantee the global optimal solution's prediction. Compared with combinatorial optimization, which can flexibly use different objective functions to optimize different web design aspects, machine learning methods often cannot achieve this degree of flexibility. Machine learning often requires complex models such as deep neural networks to learn and predict user interfaces with complex structures. However, such models are often poorly interpretable. Hence, they are called "black-box models", which means that it is difficult for developers to explain the internal mechanism of the model and why the model can give such results. Finally, a single model's training often requires much higher computing time and computing resources than combinatorial optimization. It will become more complicated when different optimization strategies need to be performed for different users, such as user interface optimization using the user's operation history as an optimization parameter.

2.4 Summary

This section first introduces the essential technologies of web user interface development: HTML, CSS, JavaScript. These technologies are closely related to the development process of C-RWD. Then, from the historical perspective of web layout design, it reviews the development and evolution of web layout design and eventually moves to responsive web design, which will be introduced as an essential topic in Section 3. Finally, it introduces computational methods from the perspective of user interface optimization and explained the role and application of computational methods in interface optimization.

3 Responsive Web Design

3.1 Overview

Responsive web design is an essential element in modern web design. At the same time, it is also one of the critical knowledge involved in C-RWD. This section will introduce the definition, essential elements, framework, and responsive web design services. Furthermore, this section will help understand responsive web design and play an essential role in understanding the C-RWD system's implementation in Section 4.

3.1.1 What is Responsive Web Design

Responsive web design was first proposed by Marcotte Ethan [32]. Its basic idea is that web pages should respond to user equipment changes, window size, user environment, and other factors. In other words, the web page design that is finally presented to the user should not be static but should be adjusted and changed according to the user's context. One of the most common examples is that a responsive web page looks different on a mobile device screen and a wider computer screen. Due to the large gap between mobile devices' narrow screen space and the larger display space of computer screens, many design elements need to be reconsidered and designed to ensure that users have a good user experience on different devices.

Responsive web design comes naturally with the evolution of the diversity of user device sizes. Since most users browse the page with computer monitors in the initial web design period, web designers only need to consider designing web pages for these desktop devices with similar screen widths. Therefore, a 960 pixels fixed-width layout with white blank areas on the left and right side is the mainstream layout design method [15]. Nowadays, users use more and more various devices. From Table 1 derived from the statistical data of [2], it can be seen that different display devices with very different resolutions constitute the global display device market share. Users may browse the same website using different devices such as mobile devices with small screen widths, computer monitors with medium screen widths, and TV devices with a large screen, in different scenarios. Therefore, responsive web design has become a design factor that web designers must consider.

Table 1: Worldwide screen resolution market share in May, 2020.

Width Range	**Total Share**	**Specific Composition**
0 - 400 px	25.59%	360x640 10.14% 375x667 4.24% 360x780 3.14% 360x760 2.85% 375x812 2.68% 360x720 2.54%
400 - 800 px	8.75%	414x896 3.62% 768x1024 2.77% 414x736 2.36%
800-1200 px	Not Listed	Not Listed
1200 - 1600 px	18.34%	1366x768 9.67% 1536x864 3.55% 1440x900 3.03% 1280x720 2.09%
1600 px above	8.35%	1920x1080 8.35%
Other	38.96%	Not Listed

3.1.2 Design Strategies

There are two main design strategies for responsive design: the mobile-first strategy and the desktop-first strategy [33]. The main difference between these two strategies lies in the direction of user interface design.

The mobile-first concept believes that the web user interface design should start with the mobile interface. The design target screen size should be changed from small to large and transformed from mobile into the tablet, and finally to desktop devices. The desktop-first concept is just the opposite, thinking that the web interface design should start from the desktop, then the tablet, and finally consider mobile devices' design.

The traditional web interface design process usually used the desktop-first strategy because website traffic mainly came from users who use desktop devices. Before smartphones were invented, the web design process had almost no interface adaptation

for mobile device design. Later, for a long time before mobile devices became popular, most users still used desktop devices to browse the website and not change their web browsing habits. This made website designers pay more attention to the desktop width and ignore or underestimate mobile devices' adaptation work. That is the reason why desktop-first was popular before. However, in recent years, mobile devices' market share has gradually caught up or even exceeded desktop devices. According to [34], the market share of mobile devices in 2020 has reached 50.88%, slightly surpassing the market share of desktop devices: 46.39%. This trend makes the mobile-first strategy popular. Designers who support mobile-first believe that mobile-first design can help designers evaluate the importance of different page content. Because mobile device interfaces are often narrow, page designers need to consider the importance of web content and streamline unimportant parts on the mobile screen. This process will make the content of the web page more concise and make reading more efficient. Besides, since Google Search adds the mobile version of the website to the search index by default, mobile-first can help web pages better rank in the search engine's displayed results than desktop-first. The desktop-first strategy supporters mainly believe that mobile interface design is often very time-consuming and consumes extra energy. The mobile-first strategy will decrease interface design and development efficiency.

Because the number of users between mobile and desktop devices is approaching, the wise reason for deciding which strategy to use should be the website's target users' usage scenarios and usage habits. If the website's primary users access and browse the web through mobile devices, then a mobile-first strategy should be used to ensure the mobile interface's high usability. If the user mainly uses the website through the desktop, then the desktop first strategy can ensure the desktop web interface's excellent effect.

3.1.3 Workflow

There is usually a particular workflow in a responsive web design and development process. This subsection will take the mobile-first as the design and development strategy, and briefly describe the workflow of responsive web design and development.

The first step in the workflow is usually to explore design requirements. In other words, it is to clarify what the functions of the website need to be and what the overall design style of the page should be. For example, a restaurant website needs to have functions such as dishes information display, online ordering service, and

restaurant information display. In the second step, having functional requirements, responsive web design usually follows the content-first theory and determine the required content form, quantity, and importance according to the interface's functions. For example, a restaurant website needs to have a dish display component, a contact information area, a navigation bar, and a restaurant logo, and so on. In the third step, after determining the interface's content, the designer needs to prototype these content components to clarify these components' concept. In the fourth step, the designer will start from the mobile device and design the interface prototypes under each interface size one by one according to the required layouts of different widths. There may be many iterations in this process to continuously improve the design of the interface prototype. In the fifth step, the developer will develop corresponding components based on the prototype. The developer will then fit these components into different static interfaces according to the designer's prototype interfaces of different sizes. In this step, the developer will check whether the final static interface completely corresponds to the designer's design prototypes. In the sixth step, the developer will modify the interface components to be flexible and integrate the interfaces under different breakpoints into one responsive interface by creating media queries and other CSS styles. Finally, the tester will perform usability testing on the user interface to find out the existing problems and provide feedback to the design and developer for iteration improvement.

The above steps are just an example to illustrate the process of responsive page development. Due to the different frameworks used and actual requirements, the process may have specific actual development changes. However, due to the responsive interface's cross-platform characteristics, multiple rounds of iteration and testing are usually required in the design and development to ensure that the final product has good usability.

3.2 Elements of a Responsive Design

The responsive web design process requires designers and developers to master many different aspects of knowledge and skills. This process usually involves different technical elements such as CSS modules, HTML structure, responsive web components design, and the overall typography. So, this subsection will introduce these essential elements for responsive web design and their applications.

3.2.1 CSS Responsive Layout Modules

Some CSS layout modules can help developers create a responsive web. For example, CSS flexible box layout [20] is a CSS module that could be used to make elements flexible. The flex container and items can shrink and grow and flow to fit different screen sizes. CSS grid layout module [22] is another module of CSS which could provide a grid layout and help the web layout organize the responsive grid layout. The developer can also achieve the element's automatic floating effect by setting the float style [35]. The float style is usually applied to some simple responsive development requirements.

3.2.2 Meta Viewport Tag

The viewport meta tag was first proposed by Apple in the Safari Development Guide [36] to help interface developers develop responsive interfaces for devices of different sizes and physical resolutions. To understand how the viewport meta tag affects responsive development, three different viewports, namely visual viewport, layout viewport, and ideal viewport, will be introduced first. They were proposed by [37, 38] and used to help understand the concept of the viewport in different situations.

The first one is the visual viewport. It represents the size of the currently visible area of the browser. It does not include the frame of the browser itself. The second viewport is the layout viewport, which is a container for HTML elements. It represents the layout composed of all the currently rendered content. The relationship between the visual viewport and the layout viewport can be understood as the browser rendering the entire page into the layout viewport. The user's current browser's visual view is like a window tool used to browse the entire page. This window tool can slide around and move closer and away from the layout viewport to see different layout viewport areas. In other words, users can browse the content of the page by zooming in and out and adjusting the visual viewport, while the layout viewport is fixed. The relationship between them is shown in Figure 9. The third viewport is the ideal viewport. This viewport appears mainly because mobile devices of the same width may have different screen resolutions. For example, the resolution width of some mobile devices equipped with a retina screen is twice the device's width represented by the CSS pixel. Therefore, a pixel unit on such a device is no longer a traditional pixel, but a device-independent pixel (DIP) composed of multiple physical pixels. The ideal viewport is a viewport that uses dots per inch (DPI) to represent the resolution of the device. DPI can facilitate the RWD adaptation of different kinds

of mobile devices. Therefore, using an ideal viewport can ensure that the interface design is perfectly adapted to a certain device.

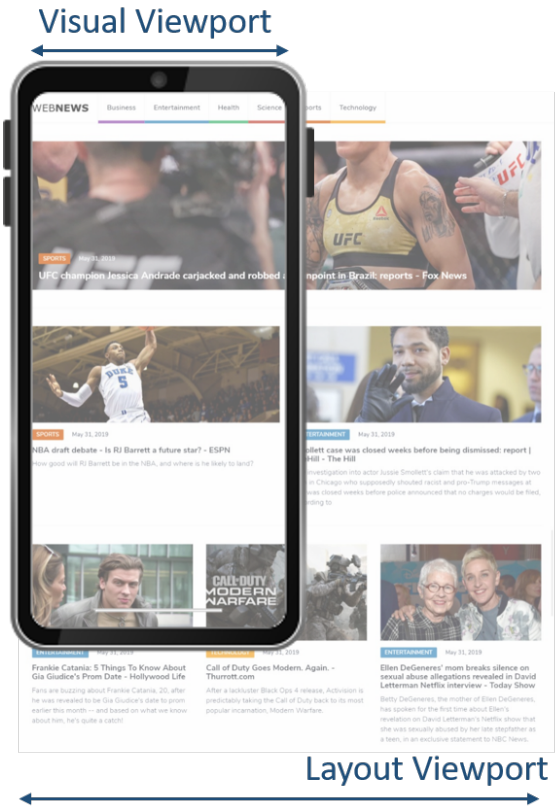


Figure 9: Visual viewport and layout viewport comparison.

When the browser of the mobile device renders the page, by default, the interface is first rendered under the layout viewport set by the browser. For example, the Chrome browser sets the layout viewport to 980px by default. However, the screen size of mobile devices is often less than 980px. For example, for a mobile device with a screen width of 320px, the browser will first render the interface at a width of 980 pixels and then reduce it to a size of 320 pixels and display it on the screen. This process is shown in Figure 10. This process causes two problems. First, for non-responsive websites, the page's font and image size will become too small, which will seriously affect the user experience. Users still need to enlarge the page to browse the web manually. Second, for responsive web pages, since the browser's default layout viewport is 980px, this will cause media queries with a width smaller than this value to lose its effect and can never be triggered. To solve this problem, the meta viewport tag needs to be used. The web developer could insert a meta tag named viewport in the page's HTML file's head element. The code is as follows:

```
1  <meta name="viewport" content="width=device-width,
       initial-scale=1,maximum-scale=1.0">
```

Here the meta tag indicates that this element is used to control and describe the HTML file. It can be parsed and recognized by the browser but will not be rendered on the interface. The `name` attribute of this element indicates the name of the metadata represented by this tag. The `content` attribute is used to describe specific information related to metadata. The `width=device-width` in the content attribute means that the device's width is used as the layout viewport's width. The attribute `initial-scale=1` means that the visual viewport ratio to the interface's ideal viewport is 1. Usually, only setting the values in these two contents can make the CSS pixel of the interface correspond to the DPI of the device. However, due to the different characteristics of different devices and browsers, web developers usually need to set these two values in content at the same time to ensure this.

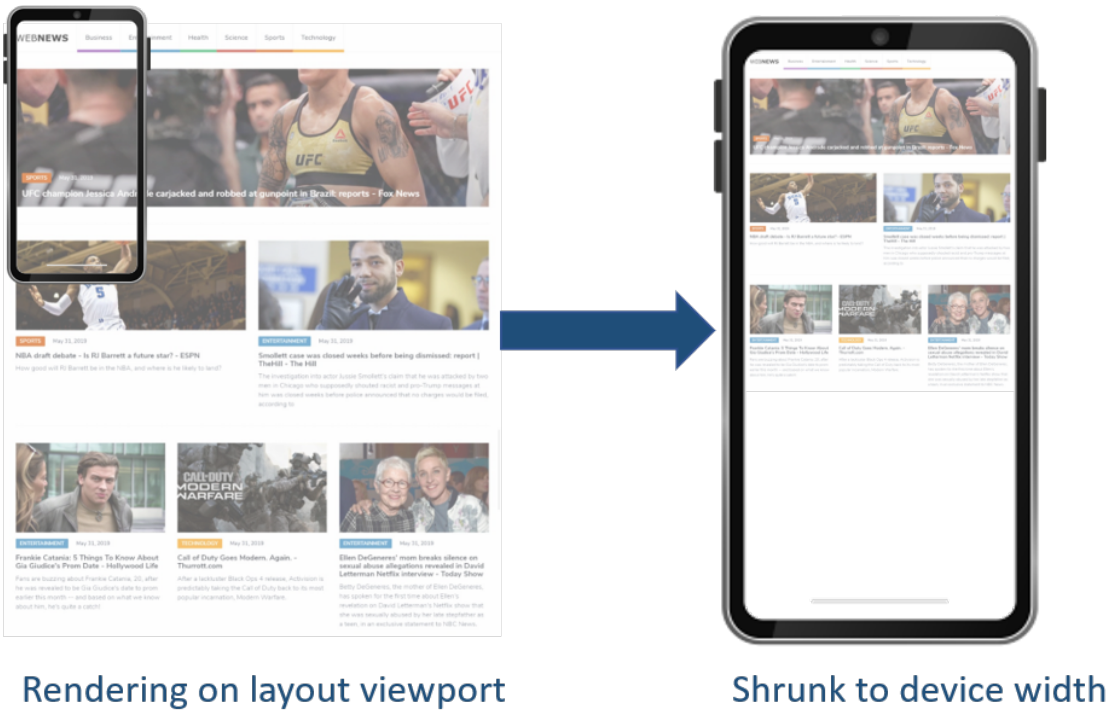


Figure 10: Layout rendering process on mobile device.

3.2.3 Media Query

To customize the style of components in different screen sizes in RWD, developers need to use media queries. A media query consists of a media type and some related expressions to check the media features. These expressions could be combined to

check complex condition [39]. In web development, commonly used media types are screen, printing, and commonly used media functions are width, orientation, and hovering function.

In responsive interface development, developers usually use media feature width to detect the interface's current width and set different styles for different width interfaces. Setting min-width and max-width to limit the different viewport size range, the style under that media query will be applied to the interface when the web page interface's size meets the specific interval. In this way, developers can design different interface layouts for interfaces of different size ranges. Media queries can be directly added to CSS, which is also the most commonly used media query method. In addition, media query can be added through JavaScript methods such as `Window.matchMedia()` and `MediaQueryList.addListener()`. The advantage of using JavaScript to add media queries is that developers can dynamically add or delete media queries to achieve more complex media query control.

3.2.4 Flexible Media

In responsive interface development, developers usually need to set various media content, such as images and videos, to be flexible. When the interface size changes, these media content's size also needs to be resized somehow.

For image, this process is relatively simple. Developers only need to set the image element's width as a percentage of the parent element, such as 100%. Then set the height of the image element to `auto`. The image element will maintain the original aspect ratio by default and change flexibly with the size of the parent element. Besides, suppose the size ratio of the image is different from that of the parent element. In that case, developers can also use the `object-fit` attribute on the parent element to adjust the image's display style. For example, by setting this attribute to `cover`, the image will be rendered at its original size. However, only the image part within the parent element's display range can be displayed in the interface. The `object-fit` attribute can also be set to `none`, `scale-down`, `fill` and `contain`. The specific effect of each value is shown in Figure 11. In Figure 11, an image with a width of 400 pixels and a height of 300 pixels is displayed in a container element 100 pixels wide and 120 pixels high using different object-fit values.

For video content, if the video is created using the `video` HTML tag, making it flexible is the same as the image element above. However, in actual development, developers often use the `iframe` element to embed a video sourced from an external

Figure 11: Difference between each object-fit value.

website. For this kind of video element, developers often need to adjust the video's size by adding a wrapper and controlling the wrapper. Usually, this wrapper element will maintain a particular aspect ratio, and the specific setting method will be described later in Section 3.2.6.

3.2.5 Responsive Media

Responsive media mainly includes responsive videos and responsive pictures. Compared with flexible media, which transforms a single media object's size, responsive media aims to change the source of the media content according to different device conditions.

For example, because different devices have different pixel per inch (PPI) values, devices with lower pixel density often only need to load relatively low-resolution images. Devices with high pixel density require high-resolution images to give full play to the device's display performance and improve the fineness of the picture display. To achieve the effect of dynamically switching the image resolution and size, the developers can use the `srcset` attribute and the `size` attribute in the image element together. The `size` attribute is used to obtain the current device status through media query. The `srcset` provides a set of image addresses to switch image resources according to the media query results. More information on how to implement responsive images can be found in [40].

For responsive video, it can be achieved by wrapping multiple `source` elements in the `video` element in HTML5. For each `source` element, developers can use the `media` attribute to clarify the usage through media query and then dynamically switch the video resource's quality.

One of the main benefits of using responsive media is that it can dynamically load resources based on the device's characteristics and internet performance. For high-resolution retina screens, responsive media can provide high-resolution pictures to improve user experience and avoid blurring pictures. Besides, the dynamic loading of

media resources can also help users save bandwidth and speed up web pages' loading speed. For example, it is meaningless to load a high-resolution picture on a device with a lower resolution. The loading process will increase the download time but will not improve the display quality of the interface.

3.2.6 Maintaining the Aspect Ratio

In developing responsive interfaces, developers often need to maintain the aspect ratio of elements such as card elements. However, the current CSS standard does not formally propose such an attribute to play this role. In the latest W3C editor's draft [41], the `aspect-ratio` attribute is proposed to maintain the aspect ratio of an element box. This new feature is still in the draft stage, and most browsers do not yet support this attribute.

Therefore, to solve this problem, developers can wrap the target element that needs to maintain the aspect ratio through a wrapper container and set relative styles on the wrapper element to maintain the aspect ratio [42]. This technique is shown in Figure 12 as a example. The aspect ratio container element will be referred to as the container element for short. The target element that needs to maintain the aspect ratio is called the target element. For the container element, its `overflow` attribute is set to `hidden` to create a new blocking formatting context (BFC) [43]. Under the new BFC, the height of child elements whose `position` attribute is `absolute` will also be counted. The `padding-top` style of the container element is set to the required aspect ratio, which is 100% in the example. The principle of this is that when the value of the `padding-top` style of an element is a percentage, the percentage is calculated relative to the element's width. So setting this style to 100% will make this container have a 1:1 aspect ratio. For the target element that is a child element of the container element, its `position` style is set to `absolute`. Its `left` and `top` styles are set to 0 and its `width` and `height` styles are set to "100%". This way, the target element will cover the entire range of the container element and follow the container element aspect ratio. The aspect ratio of the target element is well maintained.

3.2.7 Menu

When switching between interface widths of different widths, one component that needs to be redesigned is the menu component. Since a website's menu is usually used as the navigation bar of the entire site, its structure is often complicated. A

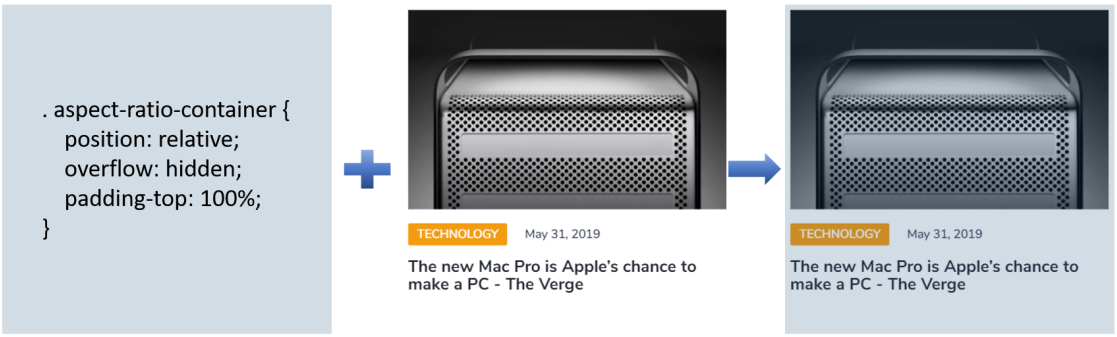


Figure 12: Process of maintaining the aspect ratio of a card element.

menu usually consists of multiple menu items, and each menu item may have its menu structure, which is called a sub-menu of the menu item. Due to the limitation of the interface width, menus of different widths need to have different designs, and media queries are used to switch these different designs with the width change.

For desktop devices, the interface's width is often enough to display all the first-level menu items. In that case, the first-level menu items can be all displayed on the menu, and the second-level menu items often need to be reached in the form of the drop-down by clicking the first-level menu item. For mobile devices with narrow screens, the first-level menu items cannot be displayed directly on the menu. The hamburger menu could be used to solve this problem. There are only website logos and a toggle button on the interface of the collapsed menu component. Users can expand the menu to view the first-level menu items by clicking this switch button. The hamburger menu can save screen space on mobile devices while maintaining all the information on the menu. This method's disadvantage is that users may need to frequently expand and close the menu to find the desired menu item. A responsive hamburger menu is shown in Figure 13.

Another widespread implementation of responsive menus on mobile device is the off-canvas menu. Generally speaking, the functions between the off-canvas menu and the traditional drop-down hamburger menu are not much different. However, when the menu has a large number of menu items, the off-canvas menu can be used to optimize the user experience of the menu on mobile devices. The idea of off-canvas is that users can make the menu enter and exit the interface from the left side of the interface by clicking the toggle button at the interface's upper left. The entire off-canvas menu interface is outside the main interface as if it is separated from the website canvas. The advantage of using the off-canvas menu is that users can swipe

up and down on this menu to find their goals from a large number of options because it is separated from the website interface. A responsive menu using off-canvas is shown in Figure 14.

In short, as an essential part of responsive web design, responsive menus require careful design by developers to improve the usability of the interface.

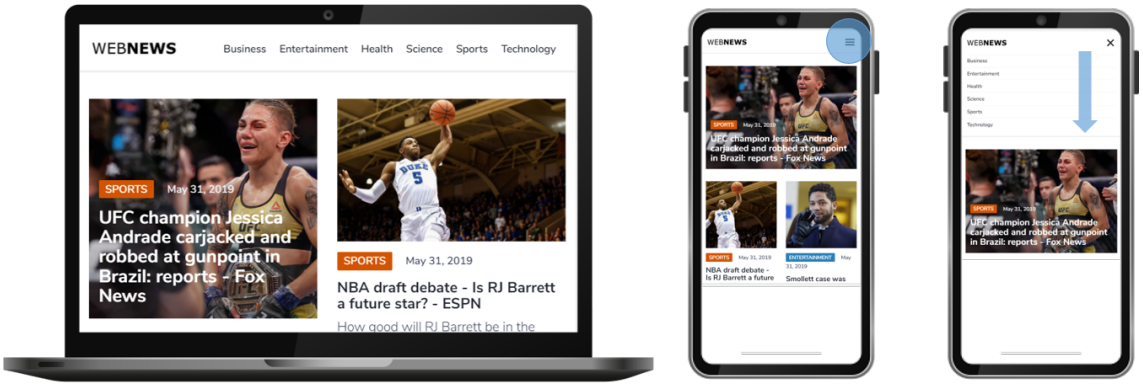


Figure 13: Hamburger menu in different states: (a) horizontal expanded state on a desktop wide screen, (b) collapsed state on a mobile device screen, and (c) expanded state on a mobile device screen.

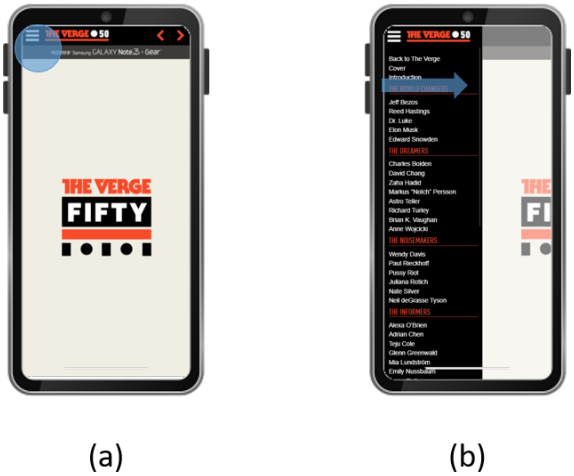


Figure 14: Off-canvas menu in different states: (a) collapsed state and (b) expanded state.

3.2.8 Typography

The typography of the interface is worth designing carefully to get excellent readability. This subsection will introduce how responsive web design should handle interface typography from two aspects: the font's responsive size and the optimal line length maintenance.

First of all, for responsive font sizes, developers often use `em` or `rem` as the unit of font size and work with media queries to make the font size change with the interface width. Compared to using absolute units such as pixels to set the font size, the advantage of using `em` or `rem` is that they are both relative or can be easily set to be relative with the font size of the root element. When users browse the website, they can adjust the page's font size by zooming in or out. The media query can detect the current interface width and switch the font size pre-defined by the developer. Besides, developers can also use the `vw` unit to set the font size to achieve fluid font. The `vw` unit is the font size unit relative to the current viewport width. One `vw` is equal to one-hundredth of the current viewport width. The advantage of using `vw` is that the font size can change with the interface width. However, the disadvantage of using `vw` is that when the interface width is extreme, the website's font size may be too large or too small. This problem can be alleviated by carefully setting the media query to switch between different font sizes. Developers can also use the CSS `calc` function to combine the `rem` unit and the `vw` unit to solve this problem completely. For example, developers can set the font size to `calc (x rem + y vw)` to make the font size the smallest x rem and avoid too small font size.

In terms of optimal line length, according to Web Content Accessibility Guidelines (WCAG) 2.1 [44], the number of characters or glyphs in a line of text should not exceed 80. The study of [45] on web readability also validates and supports the value of no more than 80 given in WCAG 2.1. Therefore, developers need to use the media query to adjust the font size and the width of the text's container element so that each line's character number is within 80. A rough method is to set the container element's width using the CSS length unit `ch`. One `ch` is equal to the "0" (ZERO, U+0030) character width under the current font size according to [46]. When the container element's width is set to 80ch or less, the text content using a fixed-width font such as Courier will remain below 80 characters per line. The text content using variable-width font styles, such as Helvetica, cannot guarantee this [47]. Therefore, developers need to carefully adjust and test the website's text elements to ensure that they have an optimal line length in the development process.

3.3 Frameworks

The responsive web development framework can help developers reduce repetitive work and improve the efficiency of web development. This subsection will introduce CSS frameworks and JavaScript frameworks related to responsive web design and briefly compare and analyze these frameworks' characteristics and application scenarios.

3.3.1 CSS Frameworks

CSS frameworks could help developers on responsive web development. They can provide a large number of ready-made responsive components such as navigation bars and forms. Besides, they can speed up the developer's website development process while ensuring the website's functionality and good usability. CSS frameworks also usually provide their responsive layout utilities to help developers implement responsive layouts.

For example, Bootstrap is a popular front-end framework that supports fast design and develops responsive web sites [48]. It provides rich RWD features by providing responsive container, grid system, and layout utility annotation. After developers have mastered Bootstrap, they can flexibly combine these responsive features to develop excellent responsive websites quickly. Besides, since Bootstrap is an open-source, customizable CSS framework, developers can customize the framework's features and styles by downloading and modifying its source code.

The advantage of Bootstrap is that it can help developers efficiently complete the development of responsive websites. However, the premise is that developers need to pay additional learning costs to master Bootstrap. Bootstrap has made the components simple and easy to use. Developers only need to change a few properties to make the component meet the functional needs. It is advantageous for junior developers who pay attention to component functions and not particularly pursue interface details. However, this can also be a disadvantage. When developers want to customize these components' styles and details deeply, they often need to spend much time modifying the source code.

In addition to Bootstrap, Foundation is also a well-known open-source CSS framework [49]. It has many similar functions with Bootstrap. Its components are not entirely ready and polished to use like Bootstrap but require a specific custom design. Therefore, it is more challenging to get started than Bootstrap. However, this also makes the website using Foundation have its unique design style. Frequently,

the websites that use Bootstrap would have some recognizable characteristics. In short, Bootstrap is suitable for the rapid development of website interfaces, while Foundation pays more attention to the personalized design and aesthetics of the website.

3.3.2 JavaScript Frameworks

JavaScript library is another type of tool which can create RWD and help responsive web development.

Masonry is a JavaScript grid layout library [50]. It works by placing elements in optimal position based on available vertical space, like a mason fitting stones in a wall. It repositions fixed-sized items based on the available vertical space according to the order. Packery is a JavaScript library that makes gap-less and draggable layouts, and it uses a bin-packing algorithm to fill in empty gaps [51].

Isotope is a JavaScript library that can sort, filter, and lay items in user's specifications on the client-side without changing HTML markup. It is best used in the scenario where users interact, for example, sort, filter, rescope with a large group of items [52]. It supports using the Masonry or Packery layout algorithm as well as other layout algorithms.

All these three JavaScript libraries can make a responsive layout using the grid system. Masonry is the first invented one, while the other two libraries are created based on Masonry with extra functions [53]. For example, sorting and filtering functions are added for Isotope, and the draggable feature is supported for Packery. They are good at display content such as images and cards. Developers like to use them, especially when developing websites like portfolio, image gallery, or dashboard, which contains massive card-style information [53].

3.4 Services

In the process of responsive web design, some services such as web builders and responsive media providers can help developers accelerate the development speed and improve the website's user experience. This subsection will introduce and analyze some commonly used services.

3.4.1 Visual Web Builders

There are various kinds of visual web builders which can help developers to develop websites. With the popularity of responsive web pages, most of these web page builders have also supported responsive web design—for example, Squarespace[54], which is a visual web builder and provides many responsive web templates for users. Users can customize the web page and put content into the template to make a new web page automatically responsive according to the predefined responsive rule. Wix [55] and WordPress [56] are also popular templates based web builders, and they work similarly with Squarespace. They all can help web owners build responsive websites. Among these three web builders, the advantage of Wix is that it is the easiest to use. WordPress is good at building highly customized websites and managing content. Squarespace is famous for providing elegantly designed templates and supporting services such as e-commerce.

In addition to traditional template-based web builders, AI-supported web development platforms are also gradually emerging. TeleportHQ [57] is an AI-supported web front-end development platform. It is committed to accelerating the process of web development through AI automated code generation. It has two main functions. First, it can use artificial intelligence algorithms to identify the designer's prototype wireframe and automatically generate the corresponding website code. Secondly, it can use AI to provide next step predictions and suggestions based on the current prototype design to help designers improve efficiency. When using it to develop a responsive interface, it still requires designers to design for interfaces of different widths. Users need to master specific HTML and CSS skills to use this platform.

3.4.2 Media Optimization

Responsive websites usually require responsive adjustments for media content. As the responsive image mentioned in Section 3.2.4, developers need to prepare multiple resolution versions of the same image and manually set their responsive switching using CSS media query. This process is often repetitive and takes up much time.

To simplify this process, Cloudinary [58] provides a web image back end service to help developers quickly set up responsive images. Developers only need to import the corresponding library and provide image resources. Then, Cloudinary can automatically generate responsive image breakpoints and corresponding resolution images as needed. Besides, Cloudinary can provide service that automatically generates videos with different resolutions. Developers only need to import the

corresponding library and provide the original video. Cloudinary can automatically analyze the video through AI and fit the video to different resolution requirements.

Another tool that can be used to optimize media content is ImageKit [59]. It can provide responsive image services. Its highlight function is that it can provide intelligent cropping of pictures based on attention and face recognition. Therefore, it can intelligently generate thumbnails of various sizes and keep most of the picture's critical information still displayed instead of cropped. Also, "smartcrop.js" [5] can be used to crop images according to available space automatically while keeping the important information. It runs at the browser-side locally and does not need server-side computing.

In short, the media optimization service can help developers complete the preparation of trivial responsive media content. It makes the responsive adjustment of the website's media content more automated and intelligent.

[5]https://github.com/jwagner/smartcrop.js

4 Design and Implementation of C-RWD

This section will introduce the design and development of the C-RWD system. It will start from the perspective of the usage scenarios of C-RWD. Subsequently, it designs the requirements of the system according to the usage scenarios. After that, it will interpret the C-RWD system's design from the perspective of the overall architecture. Afterward, this section will introduce the various aspects of C-RWD system design and implementation in detail. Finally, the C-RWD system's requirements will be reviewed to check whether all requirements have been met to support all usage scenarios.

4.1 Usage Scenarios

This subsection will explore the typical usage scenarios of C-RWD. Usage scenarios usually describe how users will use a product to achieve their goals. These usage scenarios will be used to design system requirements in the next subsection. The usage scenarios are conducted by analyzing users' initial website interface when using the C-RWD combined with the RQs. The specific usage scenarios are shown in Table 2.

Table 2: Usage scenarios for C-RWD.

ID	**Usage Scenarios**
S1	**The user has a fully static mobile website interface as the initial interface and wants to use C-RWD to automatically convert it to be responsive to save time.** This usage scenario usually occurs when the user already has a static mobile website or wants to build a responsive website from scratch but does not want to spend too much time on RWD. In this scenario, the user only needs to build a static website on mobile screen width which is usually easy for user to design and develop. Also, mobile first strategy is followed in this process. So, a fully static mobile website is a smart start point for C-RWD to create a responsive website from that.

S2	**The user has a fully static desktop website interface as the initial interface and wants to use C-RWD to automatically convert it to be fully responsive and save development time.** This usage scenario usually occurs when the user already has a static desktop website or only want to develop a static desktop version website. The existing old non-responsive website usually only has a desktop version and does not support a good user experience in mobile device. With the similar reasons in S1, users do not want to spend much time developing this website to be responsive. So, users will use C-RWD to make the website responsive.
S3	**The user has a semi-static mobile website interface as the initial interface and wants to use C-RWD to automatically convert it to be fully responsive and save development time.** In this scenario, the user already has or would like to develop a semi-static mobile website, which means that the website is overall static but with limited responsive components e.g., navigation bar. This situation will happen when the user has specific RWD development skills and wants to manually develop some key components such as the navigation bar as responsive. At the same time, users do not want to design the desktop width version of the interface and spend too much time on the page's overall layout to make it responsive. Therefore, users want to use C-RWD to convert the static part of the website into responsive while controlling the responsiveness of the interface's already responsive components.

S4	**The user has a semi-static desktop website as the initial interface and wants to use C-RWD to convert it to be fully responsive and save development time.** In this scenario, the user already have or would like to develop a semi-static desktop website as the start interface. The semi-static desktop website means that this is a desktop screen width interface having a static layout while some of the components like navigation bar is responsive. Like the S3, the user does not want to waste time on design the mobile version of the interface and make it responsive.
S5	**The user already has a responsive website interface as the initial interface but would like to improve the responsiveness at different breakpoints using C-RWD.** In some cases, users already have a responsive interface, but he wants to improve the responsiveness of the interface. For example, the distribution of screen sizes of website users' devices may continue to change over time. To make better responsiveness, the website owner needs to adjust the breakpoints of the responsive layout over time. Therefore, C-RWD users want to use C-RWD to improve interface responsiveness and automatically adjust breakpoints to generate personalized responsive layout.
S6	**When the user use C-RWD to convert a interface to be responsive or improve responsiveness, he would like to optimize the interface layout based on some design objectives and web user interaction history to get a better interface usability.** In this scenario, the user wants to make the interface layout optimized based on some design objectives while the C-RWD generating the responsive layout. The user wants to make the layout individual level optimized based on web users historical using record. For example, the user wants to minimize the element selection time as a design objective of the interface.

4.2 Requirements

This subsection will design and analyze the requirements of the C-RWD system. In software development, requirement engineering is an essential part. It is usually conducted by going through requirements elicitation, requirements specification, and requirements validations iteratively [60]. According to [60], software requirements are usually divided into functional requirements and non-functional requirements. This subsection will start from the usage scenarios obtained in Section 4.1 to analyze and clarify C-RWD's functional requirements. Since C-RWD is an experimental software service rather than a mature commercial software, this subsection will mainly discuss C-RWD's functional requirements and briefly analyze the non-functional requirements of C-RWD.

The requirements of C-RWD are listed in Table 3. These requirements will be revisited and validated at the end of Section 4.

Table 3: Requirements for C-RWD.

ID	**Requirements**
R1	**C-RWD must be able to convert an existing web interface design to a be fully responsive, regardless of its initial size (e.g., mobile or desktop) or design (e.g., fixed or responsive).** This requirement corresponds to S1,S2,S3 and S4. When the user's initial interface is entirely static and all elements are only designed under the mobile or desktop screen width. C-RWD needs to create responsive components such as a responsive hamburger menu and footer as needed and convert the entire layout into a responsive layout. When the user's initial interface is a semi-static mobile or desktop version, C-RWD needs to ensure that its responsive part retains its original responsive design or converted according to the user's settings. Simultaneously, C-RWD needs to convert the other parts of the interface, such as the main part layout, to be responsive. When the user's initial interface is responsive, C-RWD must be able to improve the interface's responsiveness such as responsive breakpoints automatically adjustment for different users.

R2	**C-RWD must be able to automatically optimize the layout at each breakpoint width for certain design objectives.** The C-RWD should have some components to optimize layout in different width based on web owners' design objectives such as selection time. The optimized result will be used as the layout in each breakpoint of the final responsive web page.
R3	**C-RWD must be able to optimize the interface on an individual level for each user.** Since each user has a different operation history on the interface, C-RWD should optimize the layout differently. C-RWD should provide individualized parameters for optimizing the interface layout based on the user's operation history. In this way, users can get personalized optimized interface.
R4	**C-RWD must be able to improve the usability of generated optimized responsive interface.** In the process of interface layout optimization and responsive adaptation, the usability of the interface should be improved. The final generated interface should not be a simple combination of optimized layout in each responsive breakpoints. There are many aspects which can be improved such as readability and image responsive cropping. For example, a text line should not be too long or too short, as this may create reading barriers for users.
R5	**C-RWD must ensure that the responsive interface converted is robust.** After C-RWD converts the interface to a responsive interface, the new interface needs to be robust. As the user adjusts the interface content such as text, the responsive interface can dynamically adjust itself to accommodate the content's change.

4.3 Overall Architecture

The previous subsection discussed the requirements for C-RWD. In this subsection, the overall architecture of C-RWD is presented. Figure 15 shows C-RWD's overall architecture from the perspective of process flow and data flow. This subsection will first provide an interpretation of the composition of this architectural diagram. Then, this subsection will explain the architecture of C-RWD from a process flow and data flow perspective.

4.3.1 Architecture Composition

Figure 15 describes the C-RWD system's architecture by using a static desktop interface as the starting interface. In the figure, the static desktop interface in the upper left corner is the system's starting point. The responsive interface in the top right corner is the end of the system. There are three colors of modules. The green module represents the module running on the client-side while the orange module represents the server-side module. The blue module represents the data that is transferred between the different modules. Also, there are two colored arrows to connect the different modules. The blue arrows indicate the direction of data flow. At the same time, the green arrows represent the direction of the system's process flow. Finally, the lowercase letters on the arrows are used to refer to each process for further explanation.

4.3.2 Process and Data Flow

The previous subsection discussed the composition of architecture diagrams. This subsection will build on that and analyze the architecture of C-RWD from a perspective of process flow and data flow. In Figure 15, the user's starting interface is a single static interface loaded with C-RWD services. Over time, the user accesses this interface multiple times and interacts with the interface. Eventually, this interface is converted by C-RWD into a responsive interface that is optimized for that user.

First, on the client-side, starting with the initial interface in the upper left corner of Figure 15, the interface that uses the C-RWD service will collect information about user interactions with the interface such as click events (process a) via the event logger module. The event logger then stores these interaction events as events data (process e) and sends it to the API server (process j). In process b, the layout parser component parses the HTML elements of the interface and generates original parsed layout data. The parser component then streamlines the original parsed layout and

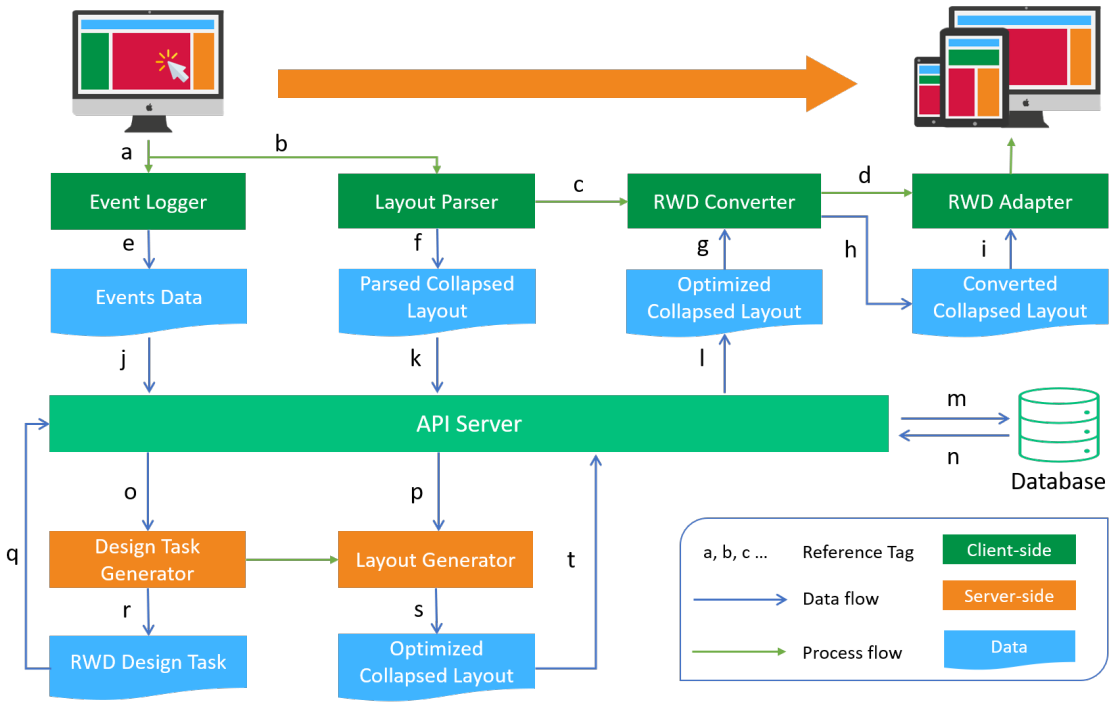


Figure 15: Architecture of C-RWD.

preserves key element information to generate parsed collapsed layout (process f) and uploads it to the API server (process k). After the API server receives the data, it stores it in the database (process m) and retrieves it from the database when needed to send it to the relevant component (process n).

Then, on the server-side, the design task generator will retrieve the layout, configuration, and event data (process o) from the API server. This data is used as input to the generator. The generator generates the RWD design task (process r) and uploads it to the API server (process q). After the design task generator has been executed, the layout generator will start running. The layout generator will retrieve the parsed collapsed layout data and the design task from the API server (process p). The layout generator will then run the optimizer and generate the optimized collapsed layout (process s) and upload it to the API server (process t).

Finally, back on the client-side, after the layout parser finishes parsing and annotating the interface, the RWD converter will be launched (process c). It will first pull the latest optimized collapsed layout data from the API server (process l) and use it as input (process g). The RWD converter converts these optimized layouts into a grid-based responsive layout: converted collapsed layout (process h). The converted layout will not be uploaded to the API server but directly to the RWD adapter

(process i). After the RWD converter finishes running, the RWD adapter will start (process d) and apply the converted collapsed layout to the web interface to complete the conversion to a responsive layout.

4.4 Client-Side Components

The previous subsection discussed the overall architecture of C-RWD. In this subsection, the components of C-RWD on the client side are analyzed in detail. The components on the client-side are the JavaScript code that runs in the user's browser. They are mainly responsible for parsing the user interface and adapting the optimized layout data. Instead of performing any time-consuming computation operations, the client-side components will send these optimization computation requests to the server-side and get the latest computation results on each load to speed up the interface's loading.

4.4.1 Component Loader

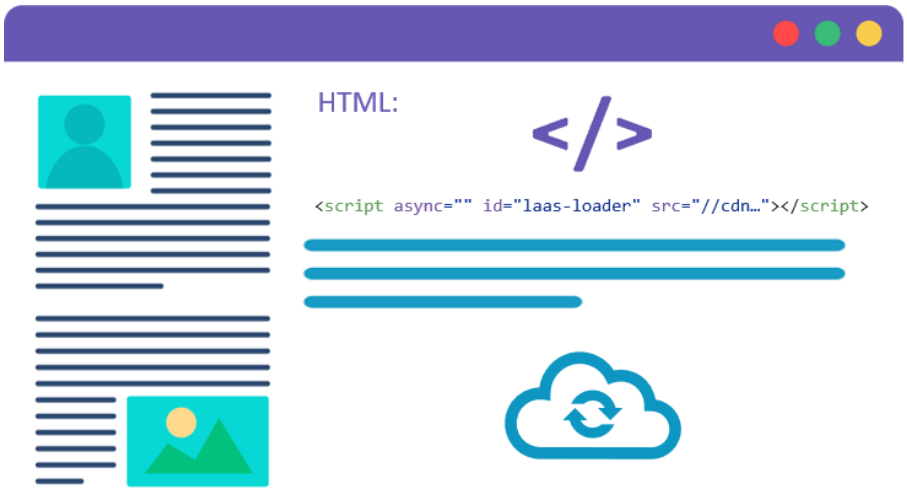
C-RWD is deployed on the LaaS platform, and users only need to introduce the loader component of the LaaS platform through a single line of code when using C-RWD. It will create a corresponding HTML element for each of the components that need to be loaded. For example, for a JavaScript type resource, it will create a script tag element and set its `src` attribute to a link to the resource on the server. After creating these elements, it inserts them at the bottom of the site's HTML document's body element. The page will load these client-side components in the final stage of loading.

The site HTML document changes before and after the loader component loading the C-RWD client-side components are shown in Figure 16.

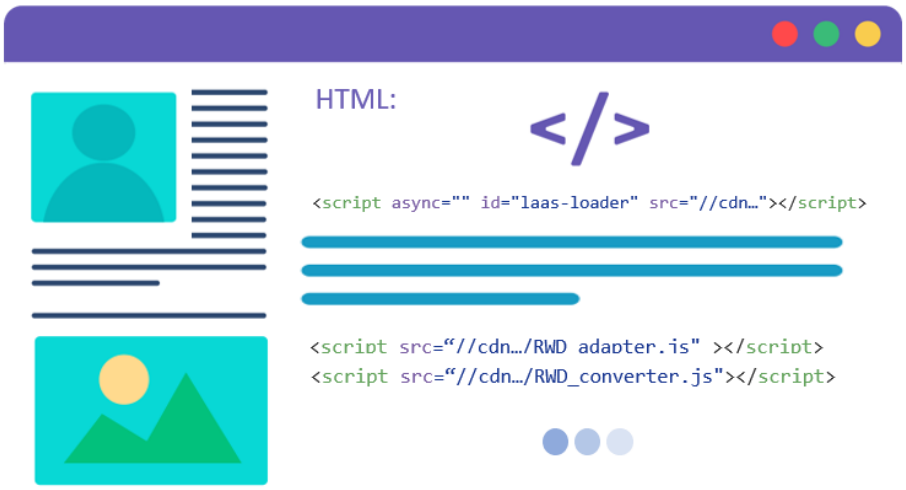
4.4.2 Layout Parser

The Layout parser component is extended from LaaS layout parser. It mainly has the following features:

1. It can traverse all web page elements and tag each element with a unique `eid`. The `eid` makes it easier for other components to locate and select an element.
2. It can detect key elements in the HTML for subsequent optimization, such as container elements and target elements.



(a) Before loading the client-side code.



(b) After loading the client-side code.

Figure 16: Dynamic loading of client-side components.

3. It can traverse the entire HTML page and parsed the properties and CSS style of each element and generate parsed original layout and parsed collapsed layout interface layout data.

4. It can generate shape sets of elements by setting different widths of elements. C-RWD adds additional filters to the process to make each shape has better usability.

The layout parser component is only executed when a user first accesses the interface.

That is, for a user, the layout parser will only start when it detects that his client id does not exist on the server. The layout parser will pull the parsed collapsed layout from the server that was uploaded the first time instead of re-generating it on subsequent user visits.

The layout parser's main working algorithm is to complete elements detection, annotation, parsing, and generating shape sets by traversing the page's DOM elements. After that, part of the information obtained will be directly set as the dataset attribute of the element and stored in HTML to be used by other components. The other part will be saved as an original parsed layout in JSON format. The original parsed layout will be filtered by the parser to generate a collapsed parsed layout, which only retains the information of the key elements. This filtering process helps reduce the transmission of redundant information and improve system efficiency. After the first run, the collapsed parsed layout and original parsed layout will be sent to the server-side database.

C-RWD has made some modifications for it to make it better support the generation of responsive interfaces. The highlighted work of C-RWD in this component is the generation of RWD shape sets of elements. In RWD, the width of the element may have a larger span. Therefore, the shape set support in LaaS may not meet this requirement well. In order to meet the optimization needs of interfaces of different sizes, C-RWD will generate shapes for each element with a width of 100 pixels, increasing by 100 pixels to a width of 2000 pixels. Besides, C-RWD will filter these shapes from multiple angles to ensure the availability of these shapes.

The key components of web pages are usually flexible rather than fixed sizes. When the element is a non-fixed size element, changing its width will make the element have a different length and width. The set of these shapes is defined as the shape set of elements. C-RWD optimizes and responsively transforms the user interface based on a grid. The layout generator's optimization for each target element is based on its limited shapes to select the appropriate size and placement in the grid layout. Therefore, the use of the shape set can reduce the layout generator's search space by trimming out poorly usable shapes in advance to improve the optimization efficiency of the interface.

In C-RWD, the `simulateRwdShapeSize` function is responsible for generating the RWD shape set of the element. Its input is an HTML element, and its output is a shape set of the element represented by a JavaScript object. Its main idea is as follows: it first creates a copy of the input element and loads it at the end of the

Body element in HTML. Then, it will initialize the style of this copy element to ensure that its width is variable. After that, it will increase by 100 pixels from 100 to 2000 pixels to change the copy element's width. After changing the width each time, it will call the internal function `shapeSetFilter` to filter each shape. The elements that pass the filter will be parsed and recorded the shape information of the element's shape, such as length and width. For elements that meet the optimal line character length, the value of its property `isIdealCandidate` will also be set to `true` to help the layout generator generate an optimized layout. Figure 17 shows the workflow of the `simulateRwdShapeSize` function.

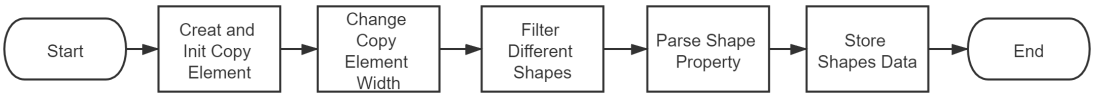


Figure 17: Working process of the `simulateRwdShapeSize` function.

The `shapeSetFilter` function has two primary uses. First, it filters out shapes with overlapping or overflow problems in internal elements. Because the optimized interface does not allow any elements to overlap each other, this will destroy the interface's usability. It will keep all other shapes. However, it will set the value of `isIdealCandidate` for shapes that do not meet the optimal line length requirements to `false`, which will make the layout generator use these shapes with low priority. These elements often have too long or too short text content in one line. And this will destroy the text readability of the interface. The `shapeSetFilter` function limits the number of characters that each line of text can have to a maximum of 80 and a minimum of 40. The reason for this setting will be explained in the following sections. For a text paragraph, when the text is more than one line, and the longest line of text strings is less than 40, it will be regarded as the text content of each line is too short. When the number of strings in the first line of text is greater than 80, it will be considered that the text content of each line is too long. Figure 18 shows how the `shapeSetFilter` function filters the shapes of elements.

4.4.3 RWD Converter

RWD converter is one of the original essential components of C-RWD. Its main idea is to convert element positions based on pixel units to the number of grid rows and columns relative to the interface container through mathematical calculations. It is mainly responsible for two functions:

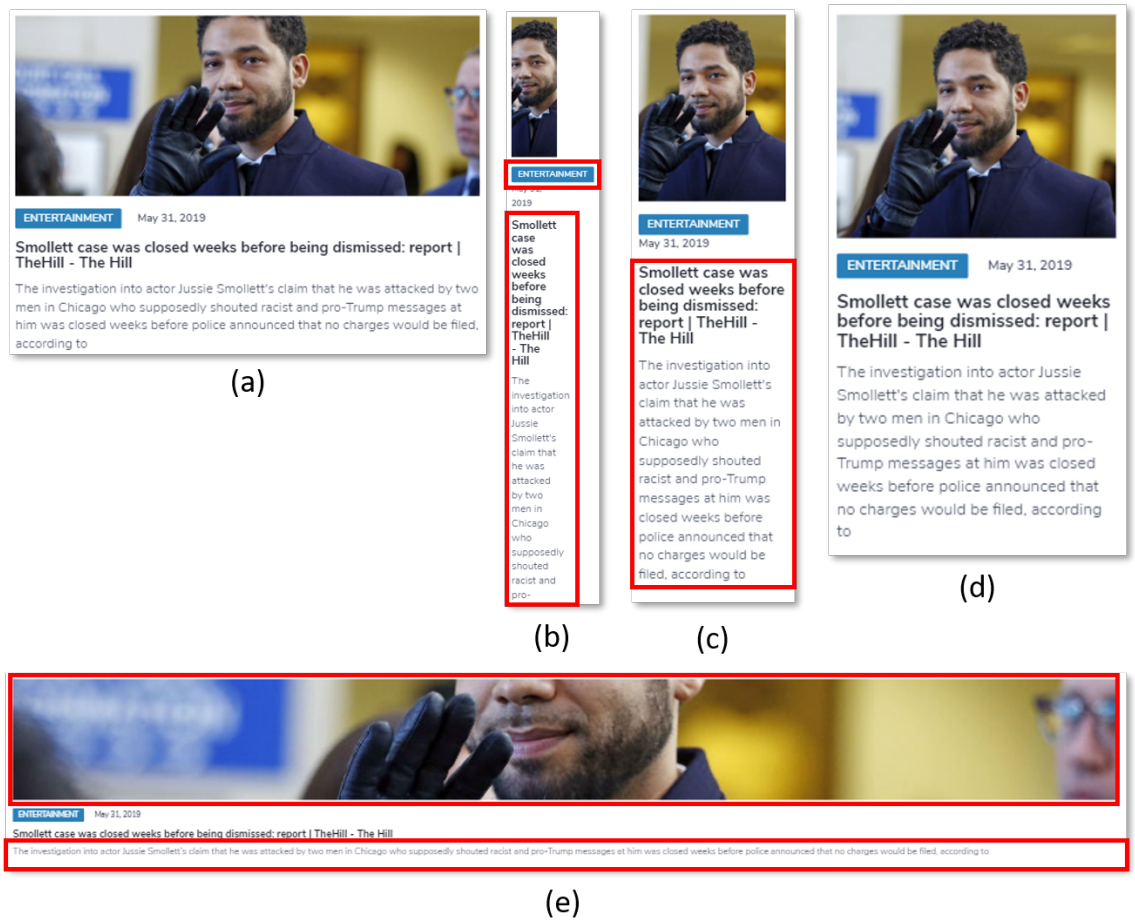


Figure 18: How shape set filtering works on different designs: (a) This is the original design with 585 px width; (b) This is a candidate design with 100 px width. There are child element overflow issue and poor readability issue. The "Entertainment" tag element is outside of the card element, and text per line is too short; The filter function will filter it out; (c) This is a candidate design with 200 px width. There is a readability issue, which is the text per line is too short. The filter function will keep this design while mark it with `isIdealCandidate` property false; (d) This is a ideal design with 300 px width, which will pass rules of the filter function and thus has a high priority to be selected in layout generator function; (e) This is a candidate design with 2000 px width. The text line is too long and the filter function will keep it but mark it with `isIdealCandidate` property false.

1. It converts the input optimized layout into a converted layout based on the CSS grid.

2. It is responsible for creating media queries for the interface. These media queries are the basis for the responsive interface to switch layouts and styles at different breakpoints.

The execution time of the RWD converter is before the interface rendering is complete. After the user accesses the interface for the first time, the server will run layout generator in need to complete the interface's optimization and save the optimized data in the collapsed optimized layout. Later, when the user revisits the interface, the server's collapsed optimized layout will be sent to the browser. When the LaaS platform detects that the `isRwd` attribute of the collapsed optimized layout is `true`, it will start the RWD converter to convert the layout data. The converted layout will be used later by the RWD adapter. Besides, the RWD converter will create media queries based on the breakpoints from the optimized layout.

The workflow of the RWD converter is shown in Figure 19. The converter will first request and obtain optimized collapsed layout data from the server. After obtaining the optimized layout, the converter will convert the position information of the elements in it from an absolute value based on pixels to a relative value based on grid rows and columns. The specific conversion process is shown in Figure 20.

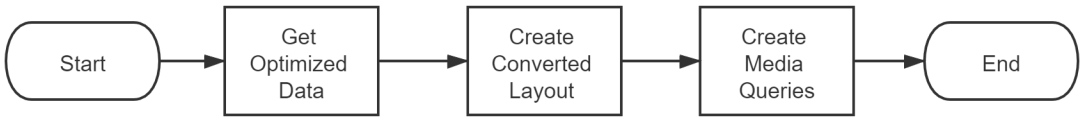


Figure 19: Working process of the RWD converter.

The converter will perform a conversion for the layout under different breakpoints. Each conversion will be carried out in two steps. The 800px wide layout in Figure 20 will be used as an example here.

In the first step, the converter will calculate the `grid-template` attribute for the layout's container element. The container element is the container of optimized target elements automatically identified by the layout parser. In a responsive grid framework, this element will also serve as the grid container. And optimized target elements will be placed in the grid container as grid items. Since the layout generator generates the optimized layout using a grid layout, and the shapes of each target element are generated in units of 100 pixels, the number of columns for the grid container can be calculated by dividing the width of the container by 100. This example layout's container width is 800px, so the number of columns in the converted grid is eight. As for the number of rows, it can be observed that each row's start line in the optimized layout is always aligned, while the end line may be different. When taking the element with the largest height in each row of elements as the

benchmark, it is easy to divide the layout into four rows. Then, for the example layout, the `grid-template-rows` attribute of the converted grid container is "24% 27% 27% 22%", and the `grid-template-columns` attribute is `repeat(8, 1fr)`.

In the second step, the converter will calculate the number of rows and columns of each element in the grid based on each element's position. Taking the "A2" element as an example, its `x`, `y`, `width`, and `height` are respectively 400, 0, 400, and 345 pixels. Then as a grid item, its transformed `grid-row-start` and `grid-row-end` attributes are 1 and 2, respectively. These values mean that it starts from the first row line and ends at the second row line. Its converted `grid-column-start` and `grid-column-end` attributes are 5 and 9, respectively. These values mean that it starts from the fifth column line and ends at the ninth column line. In this way, after all the elements are converted, the generated grid information will be saved in the converted collapsed layout for use by the RWD adapter.

After creating converted layout, converter will create media queries using `matchMedia` API of JavaScript. It will loop all the breakpoints and add the corresponding media query for each breakpoint. Because it creates media queries dynamically, users can customize the interface's breakpoints to improve the responsiveness of the interface. In other words, the breakpoint in C-RWD is entirely configurable. When the user interface's width enters the range of a specific media query, the `change` event of this media query will be triggered to call the RWD adapter's relevant functions to change the layout and style of the interface. This process will be described in detail in the next subsection.

4.4.4 RWD Adapter

RWD adapter is another original essential component in C-RWD. Its main idea is to responsively apply the converted layout data to layouts under different breakpoints by manipulating elements style. Its main functions are as follows:

1. It can select the elements that need to be modified in advance and store them in an array to facilitate subsequent adaptation operations on these elements.
2. It can perform preliminary adaptation operations on the interface that do not require converted layout data, such as excluding the influence of irrelevant intermediate elements in the element hierarchy on the grid layout.
3. It can adapt the web layout according to the converted layout. It will modify the element's style to complete the responsive transformation while retaining

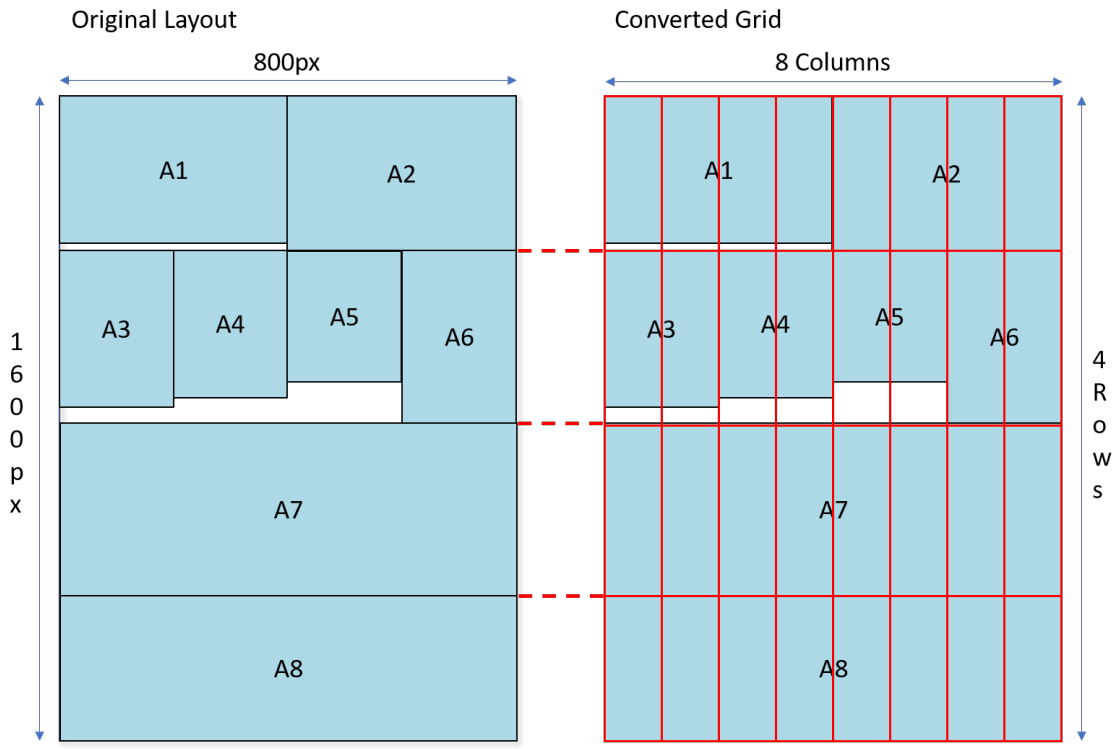


Figure 20: The process of creating a grid from an optimized layout.

the original DOM structure of the interface.

The RWD adapter's running time is after the RWD converter has finished running, and a converted layout has been generated. The RWD adapter will immediately perform adaptation operations to convert the page into a responsive interface according to the breakpoint where the interface is. When the page is adapted, it will be rendered by the browser and used by the user. When the user uses the interface, if he adjusts the interface viewport's width and the adjusted width enters another breakpoint, the RWD adapter will be called again to apply the converted layout under the new breakpoint. In this way, the webpage will have an optimized layout under different widths.

Figure 21 shows the operating process of the RWD adapter. In the first stage of the adapter operation, it will traverse all the HTML elements of the interface to find the elements used in subsequent adaptations. It will save these elements in an array to avoid repeatedly traversing the DOM tree and affecting the efficiency of the adapter function. In this process, three types of elements will be selected and saved. The first type of element is the optimized target element in the layout generator. These elements are used as grid items in the grid layout. The second type of element is

the image element contained in the target element. These elements need to perform additional operations on them during the adaptation process for smart cropping. The third type of element is the text element in the HTML document. These elements require additional operations during the adaptation process to obtain a responsive text font size.

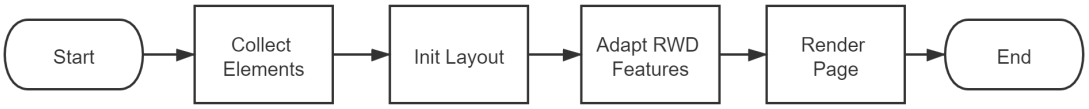


Figure 21: Working process of the RWD adapter.

In the second stage of the adapter operation, it will perform preliminary adaptation operations on the interface. It will first set the content attribute of the meta element named "viewport" in the HTML to the value required by the responsive interface. The code of this element needs to be set as the same as provided in Section 3.2.2. This change will ensure that the media query will be triggered correctly when the interface is on different devices. Then, it will perform preliminary adaptation operations on the HTML elements of the interface. The adaptation process will involve the following operations:

1. For container elements, the adapter will set their `display` style to the `grid`. Simultaneously, the style related to the size of the element will be set to `auto`. The size limit styles such as `maxWidth` and `minWidth` of the element will be set to `none`. In this way, the container element becomes a grid container whose size changes with the content. When the length of its content changes, it will automatically adjust the size to adapt to the content's length to avoid content overflow.

2. The adapter will set its `width` style to "100%" for the target element. At the same time, the adapter will set its `minWidth`, `maxWidth`, and other size limit styles to `none`. In this way, it will become a grid item element. And its width will cover the entire grid space to which it belongs.

3. For the target element's non-target sibling elements and the elements located between the container element and the target element level, the adapter will set their `display` style to `contents`. In this way, they will still exist in the DOM structure, but will not be rendered or used as grid items or grid containers. The main reason for this is that the CSS grid will treat all direct child elements

of the grid container element as its grid item. In original HTML, there may be intermediate elements between the container element and the target element. At the same time, non-target adjacent elements of target elements should not be treated as grid items.

The third stage of the adapter operation will adapt the interface responsive features according to the converted layout and the current breakpoint of the interface. The adapter needs to deal with six different types of elements, namely: container, target, header, footer, image, text elements. For different types of elements, it will have different processing methods and processing timings. These elements will be adapted for the first time by the adapter before the page is loaded. However, different elements may require further adaptation operations as the user resizes the page after the page is loaded. The timing when the adapter processes these different elements is shown in Table 4. In the adapter's first adaptation, the order in which these elements are processed is the header, footer, container, target element, image, text.

Table 4: Adaptation timings for different types of elements.

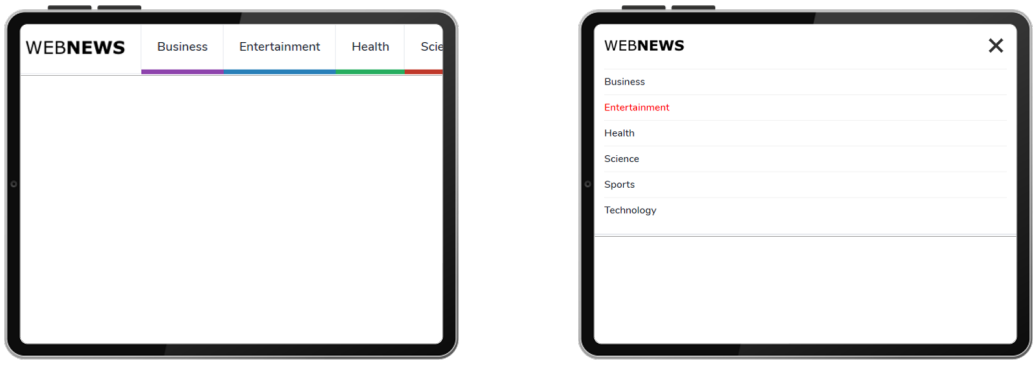
Timings / Elements	Before the page loaded	Page resized and media query triggered	Page resized but media query not triggered
Header	Yes (optional)	No	No
Footer	Yes (optional)	No	No
Container	Yes	Yes	No
Target	Yes	Yes	No
Image	Yes	Yes	Yes
Text	Yes	Yes	No

For the header element, the adapter will decide whether to perform RWD adaptation based on the layout parser's analysis result. When the original input interface of C-RWD is entirely static, all elements in the interface, including header elements, are also static. At this time, the layout parser will set the `isStaticHeader` variable to `true`. The adapter will perform adaptation operations on it. Otherwise, when the

input interface of C-RWD is semi-static, the header of this interface is responsive. At this time, the adapter will retain this header's responsive characteristics and skip the adaptation to it. The adaptation process of the header element is divided into two steps. In the first step, the adapter will extract information from the original header. It will get the original header element information such as menu logo, menu item, links. In the second step, the adapter will regenerate a new responsive header based on the acquired information. The original header and adapted header are shown in Figure 22. For this header, the information parsed and obtained by the adapter is as follows.

```
{
  "title": "WebNews",
  "logo": {
    "href": ".../index.html",
    "src": ".../logo.png"
  },
  "background": "rgba...",
  "menu": [
    {
      "name": "Business",
      "href": ".../business/",
      "font": "600 16px...",
      "color": "rgb..."
    },
    {
      "name": "Entertainment",
      "href": ".../entertainment/",
      "font": "600 16px...",
      "color": "rgb..."
    },
  ...
    {
      "name": "Technology",
      "href": ".../technology/",
      "font": "600 16px...",
      "color": "rgb..."
    }
  ]
}
```

For the footer element, the adapter will also adapt it according to the layout parser's analysis result. When C-RWD's input interface is completely static, the adapter will



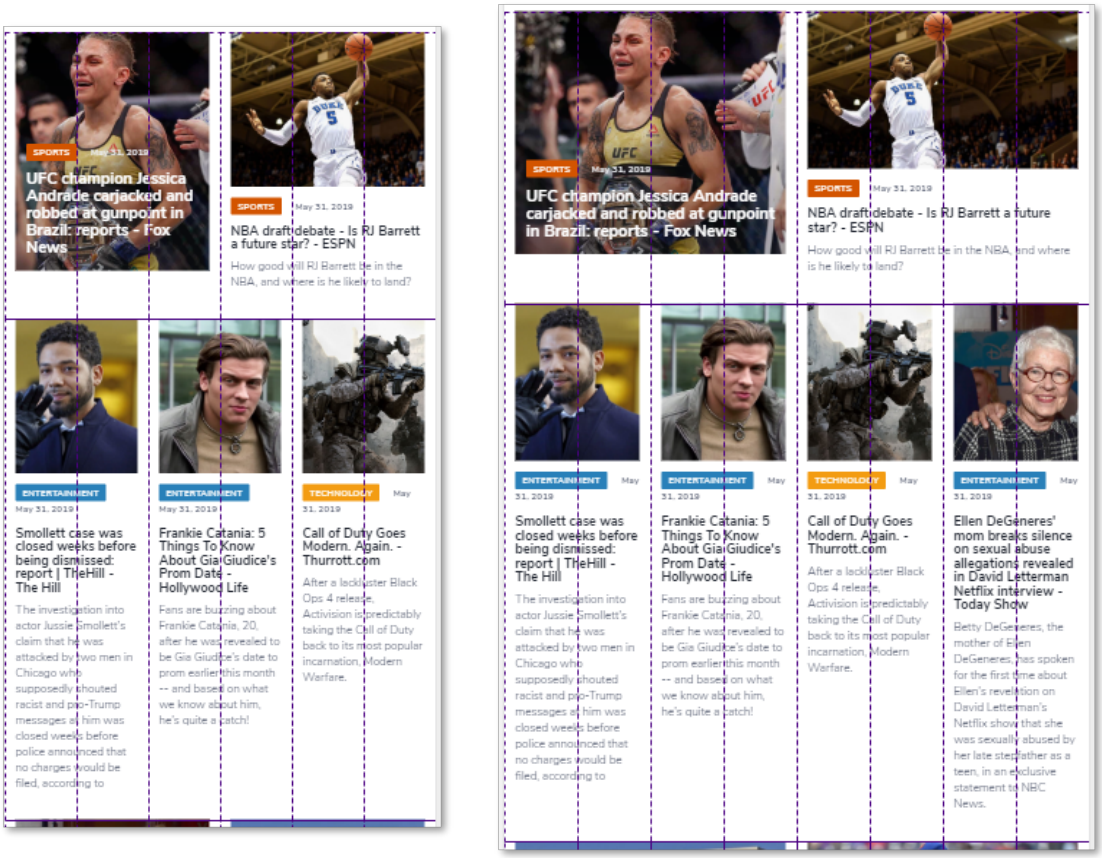
(a) Original static header. (b) Adapted responsive header.

Figure 22: Comparison between the original header and adapted header.

process the footer element. Otherwise, when the footer element itself is responsive, the adapter will skip the adaptation to it. The adaptation process of the adapter to the footer element is as follows. First, the adapter will set the `display`, `flex-direction` and `flex-wrap` styles of the footer's container element to `flex`, `row` and `wrap` respectively. In this way, this element will become a flex container. The adapter will then set the `flex` attribute of the child element in the footer to `1 1 auto`. In this way, when the width of the footer element changes, its child elements will flexibly adapt to the footer's width.

For target and container elements, the RWD adapter will process them two times to update the grid layout. The first time is before the page load is complete. The RWD adapter will adapt the interface's layout and style according to the breakpoint corresponding to the interface width. The second time is when the interface's width is adjusted, and the media query condition is triggered. The RWD adapter will adapt the interface's layout and style according to the interface breakpoint provided by the media query. The process of these two treatments is the same. First, the adapter will obtain the relevant data of the target elements and container element under the current breakpoint from the converted layout. According to these data, the RWD adapter will then adjust the `grid-template` attribute of the grid container element to regenerate different grid layouts. After that, the RWD adapter will use the previously saved array of target elements to assign the `grid-row` and `grid-column` styles of each target element according to the obtained data. In this way, the position of each target element in the new grid will be updated. The web page grid layouts in 600 pixels width and 800 pixels width are compared in Figure 23.

For the image element, the RWD adapter will adapt it on two occasions. The first

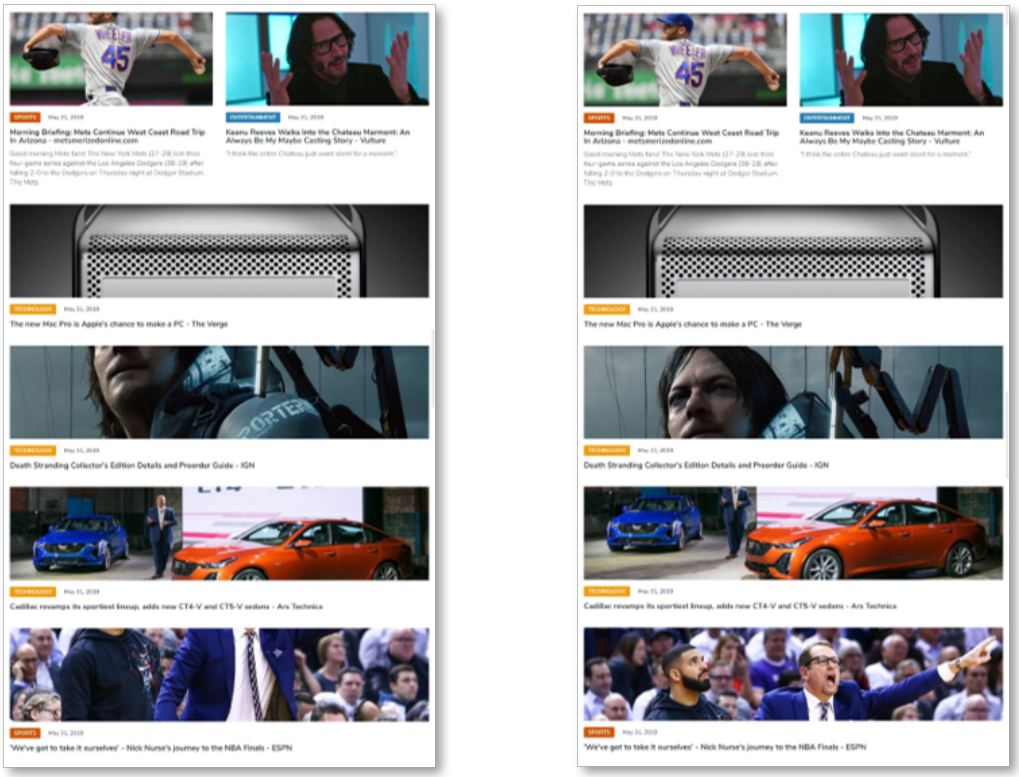


(a) 600 pixels width grid layout.

(b) 800 pixels width grid layout.

Figure 23: Different width of RWD adaptation result grid layouts.

time is before the page loading is completed, and after the header, footer, container, and target element adaptation is completed. The second time is after the user adjusts the size of the interface. Regardless of whether the media query is triggered, the adapter will adapt the image element's image. When the width of the interface changes, because the image element's width has also changed, the adapter needs to cut the image of the image element to a certain extent. In this process, the image's key information, such as the human face, will be retained in the display area. The adapter will analyze the saliency distribution in the image in real-time to crop the image. The adapter will adjust the image display area's position by adjusting the `object-position` style of the image element. Figure 24 shows the interface comparison before and after the intelligent cropping of image elements. It can be observed that the display area of the picture has been significantly optimized after the adjustment of the adapter. However, some image display areas are still not very satisfactory, such as in the penultimate image element. But this is a trade-off after balancing the time required for image analysis and the real-time adaptation.



(a) Before adapter smart cropping.

(b) After adapter smart cropping.

Figure 24: The RWD adapter's smart cropping on image elements.

Finally, for the text element, the RWD adapter processes it in two timings. The first one is before the interface is loaded. After the adapter finishes processing other elements, it will responsively adjust the text on the interface. The second adaptation timing is after the page is resized and the page width change triggers media queries. The adaptation uses CSS `calc` to make the font element calculates the font size in real time. The adapter builds the fluid font size conversion formula based on the tutorial [6] and uses media queries to apply the formula and make the font size responsive. This formula can calculate the font size based on values such as current viewport width, current breakpoint value, neighbor breakpoint value. The formula can be expressed in equation 1. In the equation, the f_{min} and f_{max} represent the minimum and maximum font size in the breakpoint range respectively. The v_{cur} represents the current viewport width which can be obtained using "100vw". The b_cur and b_next represent current breakpoint value and the next larger breakpoint respectively. By using this formula and media queries, the adapter will convert the text element's font size from an absolute value in pixels to a `calc` value relative to

[6] https://www.madebymike.com.au/writing/fluid-type-calc-examples/

the width of the viewport. In this way, when the width of the interface changes, the font size will also change responsively to adapt to the interface change.

$$F = f_{min} + ((f_{max} - f_{min}) * ((v_{cur} - b_{cur})/(b_{next} - b_{cur})) \quad (1)$$

4.4.5 Event Logger

The event logger component is extended from LaaS event logger to support more events and more features. It is mainly responsible for collecting the interaction history records of interface users to the interface, such as click events. For each interactive event, the event logger will record the event's time and specific information for the event type. For example, it will record the id of the element that was clicked for a click event. The event logger starts to listen, and log events after other client-side C-RWD components are executed before the browser renders the interface. It sends the recorded events to the server.

C-RWD implements the following new functions and features based on the LaaS event logger. First of all, C-RWD implements more events support including `laasshow`, `scroll` and `page unload` events. For example, C-RWD adds a new custom event called `laasshow` to the event logger. The `laasshow` event is triggered after the C-RWD code is loaded and before the page is rendered. It will record information such as the device type and screen size currently used by the user. This information will be used by the design task generator to adjust the breakpoints of the response interface. Secondly, C-RWD provides mobile cross-browser support for event logger. For example, C-RWD integrates different mobile interface events such as `pageShow` and `pageHidden` so that the event logger can successfully collect the interface's visibility status information under different browsers of the mobile device. Finally, C-RWD uses the `sendBeacon` API to enable the event logger to send events to the server after the user closes or leaves the interface. In LaaS, users need to wait for the event logger to send event information to the server before leaving the interface. This improvement allows users to immediately close the interface without waiting, which optimizes the user experience.

4.5 Server-Side Components

In the previous subsection, the components on the client-side were introduced in detail. In this subsection, the server-side component will be explained. These components run in a server environment and are developed using Python 3. They

start regularly to optimize and update the interface according to the user's interface optimization frequency. In other words, these components will periodically optimize the interface based on the user's recent interaction history.

4.5.1 Design Task Generator

Design task generator (DTG) is extended from LaaS DTG to support responsive design task. It is responsible for generating design tasks for the interface, which will be used by the layout generator to optimize the interface. The design task is used to express the interface's design requirements in the form of data and pass it to the layout generator. In the traditional interface design process, designers need to provide ideas for interface design, such as sorting different elements in the interface according to their importance. In C-RWD, the user's interaction data, design objectives and constraints will replace the designer to generate the design task. When the server starts to optimize the interface, it will first obtain the layout, configuration, and events data of the current interface from the database, and then pass these data into the DTG to generate the corresponding design task. These design tasks will be used by the layout generator to optimize the interface.

C-RWD's DTG has made the following key improvements based on LaaS to support the generation of responsive design tasks. First of all, C-RWD adds new attributes to DTG, such as `rwdBreakpoints` and `isRwd` to support responsive design tasks. The breakpoints of the responsive interface will be stored in the `rwdBreakpoints` attribute. Secondly, the improved DTG of C-RWD can dynamically adjust the interface's breakpoints according to the type and width of the device used by the user prompted in the `laasshow` event. For example, for mobile devices, the layout generator will only optimize its two viewport widths in landscape mode and portrait mode. Because on mobile devices, the interface can only be one of these two widths. Finally, C-RWD adds a new design objective, similarity, to DTG to better control the responsive interface's optimization through design tasks.

The DTG will generate the design task by analyzing the input layout, configuration, and event analytics data. The design task will contain all optimized target elements and their related attributes, such as `importance`, `useRwdShapeSet`. The DTG will adjust the `importance` value of the target element based on events logged by the event logger. In other words, if the user clicks on an element the most times, this element has the highest importance to the user. Then, it should have a higher weight in the optimization process of layout generating. Eventually, this element will appear

in the user interface's front position, and its size will also be appropriately adjusted to increase its saliency. Besides, the design task will also describe the optimization objectives of the layout generator. In the design task, the value of the `objectives` attribute is the current design task's optimized objectives. This value is composed of three attributes: `fitts`, `saliency` and `similarity`. The value of each attribute is a decimal between zero and one. The layout generator will read the values of these different optimization objectives during the interface optimization process to determine the combined optimization strategy. An example of a design task is attached in the appendix.

4.5.2 Layout Generator

The layout generator is extended from LaaS to support responsive layout generation. The Layout generator will optimize the interface according to the design task provided by DTG. Therefore, it runs after DTG. Its main idea is to optimize the input interface for different users by using combinatorial optimization. After running, it will save the optimized collapsed layout in the server for the RWD converter to use.

The layout generator will optimize the interface from many aspects. First, the Layout generator will ensure that there is no overlap between elements in the generated optimized interface. Simultaneously, the layout generator will also consider the aesthetics of the interface. For example, it will ensure a good alignment relationship between elements. Most importantly, the layout generator will optimize the design objectives proposed in the design task. There are three optimization objectives. The visual saliency objective will ensure that important elements have better visual attention-grabbing performance in the user interface [61]. For each element, the saliency can be calculated using specific saliency model. The selection time objective use Fitts' law to make the important elements in the interface be placed in a proper position so that user spend less time to select them. It is a common way to use Fitts' law as objective function for selection time in user interface optimization [62]. When the cursor is moved from a start point to the target element, the selection time depends on the moving distance and the target element width. In layout generator, the start point is assumed as the left top corner of the user interface. The similarity objective will make the optimized layouts have a balance between layouts similarity and individual best optimized. Combining these optimization objectives, the layout generator can generate a personalized, optimized interface for the user to help him reduce the time it takes to find and click on the elements in the interface. In this

way, the interface will have better usability. The layout generator optimizes the interface based on the grid. It uses the elements' shape set to select the appropriate element size and place them in the grid to complete the optimization. Compared to using pixel units to determine the size and location of elements, this dramatically reduces the search space and improves the optimization speed. Therefore, the layout generator can generate the optimized result of the interface within an acceptable time. More description about the principle and technical details of the layout generator can be found in [6].

C-RWD's highlight improvement work for the layout generator is that it enables the layout generator to increase the optimization support for the responsive interface. When the layout generator detects that the `isRwd` attribute of the input interface is `true`, it will enter the C-RWD mode to complete the responsive interface's optimization. This responsive layout support is reflected in the following three aspects.

In the optimization preparation stage, C-RWD added support for the RWD shape set in the layout parser component. The layout generator will use the RWD shape set to select appropriate element shapes for interfaces of different widths in RWD. In this way, in the optimization process, the layout generator will preferentially use shapes with the `isIdealCandidate` attribute of `true` to improve the interface's usability.

Second, in terms of the optimization process, C-RWD adds a new objective called similarity to the design task. Since the responsive interface will switch the layout at the breakpoint, when the gap between the two layouts before and after the optimization is too large, this will increase the user's memory burden on the interface elements' location, resulting in a bad user experience. Besides, when the user adjusts the interface's size to trigger interface switching, the inconsistency of the position of the same element will cause the user to feel confused. Therefore, the similarity objective will control target elements order difference between the original layout and optimized layout at each breakpoint. Similarity value ranges from 0 to 1. And the final elements order is sorted according to the result calculated by the Equation 2. When the value is 0, the layout generator will treat the interfaces at different breakpoints as independent interfaces to optimize them. The original layout will not affect the optimization of other layouts. When the similarity value is 1, the layout generator will make the order of the elements in the interface at different breakpoints entirely same with the original layout while ignoring the influence of other objectives. When similarity is between 0 and 1, the layout generator will take

into account both the elements importance value and the similarity objective value and give a balanced order. In the output interface, each element is sorted according to the value of $O^i_{optimized}$. In the Equation 2, $O^i_{optimized}$ represents the sorting basis value of the i-th element. $O^i_{original}$ represents the normalized value of the original order of the i-th element, which is a decimal between zero and one. S represents the current optimized similarity objective value. I^i represents the importance value of the i-th element, which is also a decimal between zero and one.

$$O^i_{optimized} = O^i_{original} * S + I^i * (1 - S) \tag{2}$$

Third, in terms of the optimized results, since the RWD page comprises multiple breakpoint interfaces, the layout generator in C-RWD mode will optimize the interface under each breakpoint in turn and merge the optimized results into one optimized layout. For example, the optimized positions of multiple breakpoints for each target element will be stored in the `rwdBreakpoints` of the element. Then, in the merged responsive layout, media queries created by RWD adapter can switch these properties for each element to achieve responsive optimized layout.

4.6 Design Choices

The design and implementation of each component of C-RWD are introduced in the previous subsection. This subsection will introduce and discuss the design choices encountered during the design and development of C-RWD. These choices affect the function and characteristics of C-RWD.

4.6.1 Layout Method

In this subsection, the choice of layout method for the C-RWD system will be discussed. In the C-RWD system design stage, which layout method to use is a crucial decision. The commonly used layout methods are grid layout, flex layout, and positional layout for responsive interface layout. C-RWD needs to choose one of these three layout methods as the layout method of the main interface.

As described in the previous sections, flex layout is a simple one-dimensional layout among the three layout methods. The advantage of the flex layout is that it can be used to create one-dimensional responsive layouts easily. However, the disadvantage of the flex layout is that it cannot support complex two-dimensional layouts. In C-RWD, the layout generated by the layout generator may be a complex two-dimensional

layout. Therefore, the flex layout cannot cope with the two-dimensional layout requirements in C-RWD.

Positional layout refers to positioning optimized elements by using absolute positioning styles. This method is used by LaaS when generating a static optimized interface. In C-RWD, the flexible interface can be achieved using the percentage unit's absolute position to layout the interface. Combined with media queries, this approach can also achieve a responsive layout. The advantage of this layout method is that it directly controls the size and position of each element in the interface through the coordinate calculation in the adaptation process. At the same time, this layout method is consistent with the method used by LaaS before. However, in actual development, the positional layout's problem is that the layout's robustness is low. Since elements that use absolute positioning are like being drawn on the interface, they have their own BFC. The browser does not perform overlap detection on them. When the content of an element changes, the positional layout is prone to problems such as element content overflow and overlap. Besides, the element's absolute positioning value is often a decimal number that needs to be rounded. It may cause unpredictable errors in the browser rounding process. Finally, in terms of cross-browser support, it has excellent browser support because it uses only basic layout features and browsers have supported this absolute position based layout method well. Because it is similar to the LaaS layout and has excellent browser support, this layout was used in the early development of C-RWD to prototype and generate some test interfaces. However, the problems mentioned above encountered during the development process indicate that it is not an ideal C-RWD layout. Therefore, despite its many advantages, the positional layout is not the ultimate choice for C-RWD.

For the grid layout, its advantage is that it can control the generated interface layout in two dimensions. Therefore, it can support complex layout requirements. Simultaneously, since the optimization logic of the layout generator itself generates the optimized layout through the grid, the RWD adapter can conveniently use this feature to generate the grid of the interface and adjust the position of the elements. Besides, compared to the positional layout, the interface generated by the grid layout has good robustness. Even if the content of the element changes, it can automatically adjust the layout to maintain good usability and avoid element overflow and overlapping. In terms of cross-browser support, different browsers have the same processing results for the grid, ensuring that the interface is consistent on different browsers. The disadvantage of the grid layout is that it cannot directly control each element's

position like a positional layout. However, it needs to calculate the number of rows and columns of each element in the grid. Since the layout generator uses the grid to generate the optimized layout, this weakness will not affect C-RWD.

Therefore, C-RWD finally uses the grid layout as the layout method. However, this does not mean that C-RWD only uses this layout. C-RWD can use flex layout on some particular components such as the navigation bar and footer to quickly generate flexible components. The difference among these three layout method is compared in Table 5.

Table 5: Comparison among three layout methods.

Layout Methods / Features	Grid Layout	Flex Layout	Positional Layout
Cross browser support	Good	Good	Excellent
Consistency with layout generator	Consistent	Not consistent	Not consistent
Element position method	Grid row and column number	Flex order and wrap direction	Absolute position based on percentage unit
Layout dimension	2 dimensions	1 dimension	2 dimensions
Layout robust	Good	Good	Bad

4.6.2 Breakpoints Selection

In C-RWD, breakpoints are a vital design consideration. The number of Breakpoints determines how many different optimized layouts the layout generator needs to generate for this website. The value of each breakpoint determines the width of each optimized layout. When users use different size devices to browse the interface, the responsive interface's breakpoint distribution will affect the user experience. For example, when the maximum breakpoint is "1200px", and the user browses the interface with a device having a larger width than this value, the website will still be rendered according to the layout at the breakpoint of "1200px". This situation may

cause the final rendered layout not to fit the current width, and cause problems such as an excessively wide width of some elements.

For breakpoints of the C-RWD system, there are three options to choose from. The first is to use fixed breakpoints settings. In other words, C-RWD uses fixed breakpoints to optimize all users and websites, which is the easiest way to implement. However, this will bring usability issues when users use a device browsing interface that differs significantly from the fixed breakpoint size. Also, different websites have different usage scenarios. For websites where the usage scenarios are mostly on large-screen displays, the breakpoints' distribution should focus on larger sizes. The second solution is to make configurable breakpoints, which means that the website owner can manually set the breakpoints of the website according to the website's design requirements and usage scenarios. This can provide specific breakpoints for each website to improve the user experience of the website. However, this solution does not consider the diversity of the user's device width. Since website developers often determine the size of breakpoints by analyzing most users' device sizes when setting up breakpoints, it will not be possible to consider the device width and usage habits of each user. When users browse websites with devices whose width deviates from the size of mainstream devices, they may still encounter usability issues. The third solution is to set the breakpoint to the device's width when the user browses the interface. The developer of the website first sets the default breakpoint of the website. Then, when the user visits this page, C-RWD will record information like device type and the device's viewport width at that time and add it to the user-specific breakpoints. For mobile devices, C-RWD will add two breakpoints to its two widths in landscape and portrait mode. In this way, the website interface's breakpoints are the result of personalized adjustments to each user.

C-RWD first uses the first scheme, which is the most straightforward scheme to implement. Then, after realizing the above limitations, C-RWD improves the system and allows developers to customize the website's breakpoints. Finally, the event logger of C-RWD is improved to support collect information such as device type, screen size and viewport width when users visit the web page. The scheme 3 is implemented. So, C-RWD's breakpoints are adjusted dynamically based on the user's interaction events. In this way, each user will have their personalized breakpoints to adapt to their own equipment and usage habits.

4.6.3 Optimal Text Line Length

The text content of responsive web pages is an essential part of the interface. For text information, an essential factor affecting readability is the number of characters per line (CPL). Therefore, the scope of the text CPL is a design choice that C-RWD needs to consider. According to [63], for English, the size of CPL should be between 45 and 75. For text content with CPL less than 40, line breaks and hyphens will frequently appear, which will destroy the user's reading experience [63]. According to the guideline from [44], to make the web page's text content have an excellent visual perception, CPL should not be greater than 80. According to the study of [45], this guideline was confirmed to have an essential connection with the text's readability.

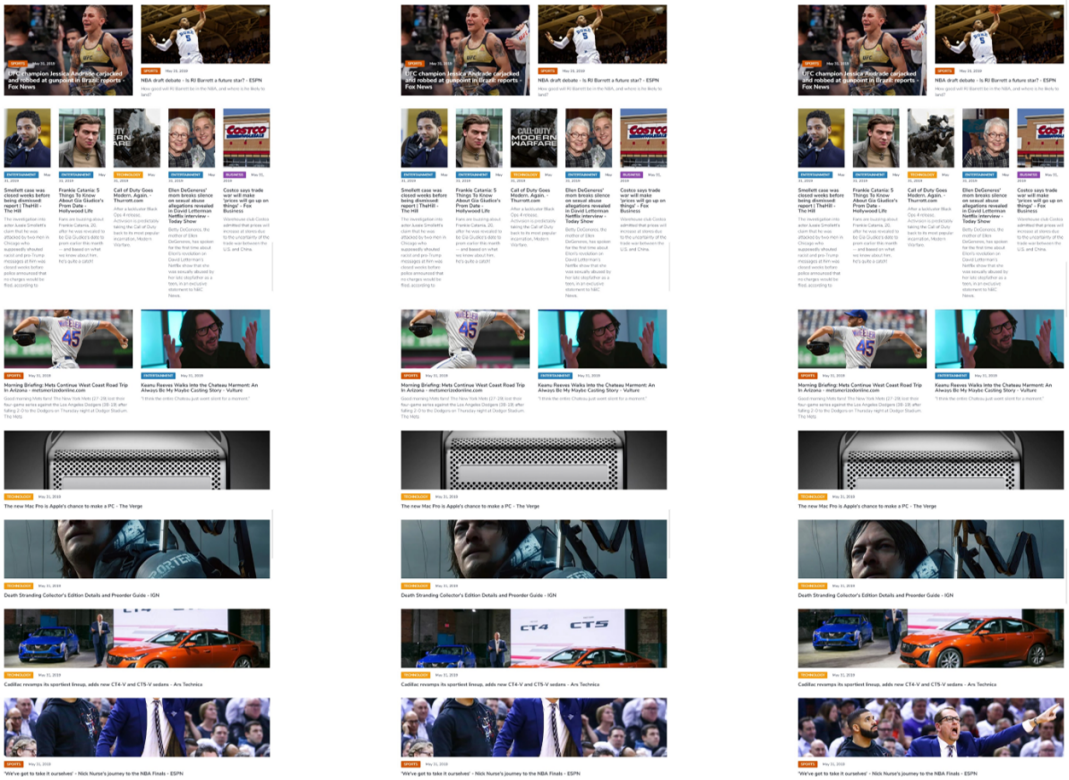
Therefore, C-RWD adds restrictions on CPL in the process of filtering shape set by layout parser. In consideration of a conservative screening strategy, C-RWD limits the CPL of text elements as follows. The CPL of the text element of the web page should be between 40 and 80. C-RWD will not restrict the lower limit of its CPL for elements with only one line of text. For multi-line text elements, C-RWD will take the longest line to detect the CPL lower limit, and take its first line to detect the CPL upper limit. Elements that pass the CPL restriction detection will be marked as ideal candidates for the layout generator to select preferentially. This less strict detection is because the text element's CPL may slightly change after the web page is resized. Using a looser selection method is conducive to providing more ideal candidates for the layout generator to improve the layout optimization's computational efficiency.

4.6.4 Image Smart Cropping Solutions

In the process of smart cropping of images with RWD adapter, which tool to use for smart cropping is a design choice that needs to be considered. There are three options to choose from. The first is that the website developer manually crops the image according to the image's content and the size of the container element. However, this will bring much repetitive work to developers. The second solution is to use ImageKit [59] to perform real-time smart cropping of images on the website. After the user resizes the website viewport, it can upload the picture and the size to be cropped to the server in real-time and perform intelligent cropping based on related computer vision algorithms. Then, the web page will reload the image to display the cropped image. Its advantage is that the calculation process takes place on the server-side, which can reduce the running time in the case of a large number of pictures. The

third solution is to use client-side libraries such as "smartcrop.js" [7] to crop the image intelligently. Its advantage is that it is not affected by the network and processes the pictures locally. It can effectively reduce the processing time when there are many pictures, and the network is slow. The interface comparison between not using smart cropping , using ImageKit and using "smartcrop.js" is shown in Figure 25.

During the development of C-RWD, the latter two schemes were used and compared. The conclusion of the comparison shows that "smartcrop.js" has a better cropping effect on the pictures used in the C-RWD test website, and the calculation speed is faster than the other one. Besides, "smartcrop.js" as open-source code can be further customized and improved, such as adding support for face recognition to improve the effect of smart cropping. Therefore, C-RWD finally chose the third option as the choice for intelligent image cropping.



(a) Original (b) ImageKit (c) Smartcrop.js

Figure 25: Comparison of different image smart cropping solutions.

[7] https://github.com/jwagner/smartcrop.js

4.7 Requirements Revisited

In the previous subsections, the design and implementation of C-RWD are introduced. In this subsection, C-RWD requirements will be evaluated to verify whether they are resolved. For each requirement, its content and the corresponding design and implementation in this section will be matched and presented in Table 6.

Table 6: Requirements and corresponding implementations for C-RWD.

Requirements	**Design and Implementation**
R1: C-RWD must be able to convert an existing web interface design to a be fully responsive, regardless of its initial size (e.g., mobile or desktop) or design (e.g., fixed or responsive).	The parser can detect whether the input web page is fully static, semi-static or responsive. For non-responsive initial interface, the layout generator and DTG will generate optimized layout in each breakpoint width. The converter and adapter will use these optimized layout to generate fully responsive layout for it while keep some original responsive features according to the user's settings. In case the responsive initial layout design, the DTG will update the breakpoints from user history visits and the layout generator will optimize the web page at updated breakpoints. The converter and adapter improve the responsiveness using techniques such as image smart cropping to generate a updated responsive web page.
R2: C-RWD must be able to automatically optimize the layout at each breakpoint width.	The parser will provide parsed layout to DTG and layout generator. The DTG and layout generator will optimize the layout for each breakpoint automatically.
R3: C-RWD must be able to optimize the interface on an individual level for each user.	The event logger will collect user interaction events. The DTG will calculate importance for each element and the layout generator will optimize and personalize layouts for each user.

R4: C-RWD must be able to improve the usability of generated optimized responsive interface.	The layout parser will filter the elements shape set according to rule such as text optimal line length, to increase the generated layout usability.
R5: C-RWD must ensure that the responsive interface converted is robust.	The grid layout used in adapter can ensure the layout could adjust itself when content change and has a good robust performance.

4.8 Summary

This section describes the design and implementation of C-RWD. This section starts with the usage scenario and analyzes the possible usage scenarios of C-RWD. Then, this section analyzes the design requirements of C-RWD based on these usage scenarios. After clarifying C-RWD requirements, the overall architecture, client-side components, and server-side components of C-RWD are explained in turn. After that, this section describes the critical design choices encountered during the design and development of C-RWD and the reasons behind these choices. Finally, this section reviews the requirements and verifies them to ensure that the C-RWD meets the design requirements.

5 Evaluation

In this section, the C-RWD system will be evaluated from multiple perspectives. First, the output of C-RWD will be displayed and analyzed to verify whether C-RWD meets the research question's requirements. Then, C-RWD will be compared with other web interface development tools. After that, C-RWD will combine the above results to summarize the shortcomings. Finally, C-RWD future work will be proposed to improve the C-RWD system further.

5.1 Output

In this subsection, the output interface of C-RWD will be displayed and analyzed. The display of the output interface will mainly use the "WebNews" website as an example. First, this subsection will show C-RWD's optimization results for different optimization objectives under the same viewport width. This will show that C-RWD can optimize the input interface based on different objectives and generate an optimized interface. Second, this subsection will show how C-RWD can generate interfaces with different viewport widths under the same optimization objectives. These interfaces will be used as the optimized interface at each breakpoint of the responsive website. This shows that C-RWD can generate interfaces at different breakpoints in a responsive interface. Later, this subsection will show the responsive interface generated by C-RWD using the optimized interface and compare the improvement of C-RWD's usability. This shows that C-RWD can improve the generated interface's usability to ensure that the final output interface has good usability. The optimized, responsive layout of C-RWD for the final output of the WebNews website will also be displayed.

5.1.1 Different Objectives

In the process of C-RWD's optimization of the input interface, a total of three optimization objectives are used to optimize the interface from different perspectives, namely selection time, visual saliency, and similarity. For each optimization objective, its optimization weight is between 0 and 1. The layout generator will optimize the interface according to the optimization weight's size, combined with these three optimization objectives. An objective's optimization weight of 0 means that this objective will not be optimized. When the objective's weight is 1, the layout generator will optimize this objective with high priority. For example, when similarity is 1, the elements' order will be completely sorted according to the original order. When the

similarity is 0, the order of the elements will only depend on its importance value and ignore the original order's influence.

The first objective is to minimize the selection time for the user to select elements in the interface. The visual saliency objective value is set to 0, and the layout similarity objective value is set to 1 to eliminate the interference of these two objectives. The value of selection time objective is set to 0, 0.5, 1.0, respectively to observe the changes of the interface layout after optimization. The result is shown in Figure 26.

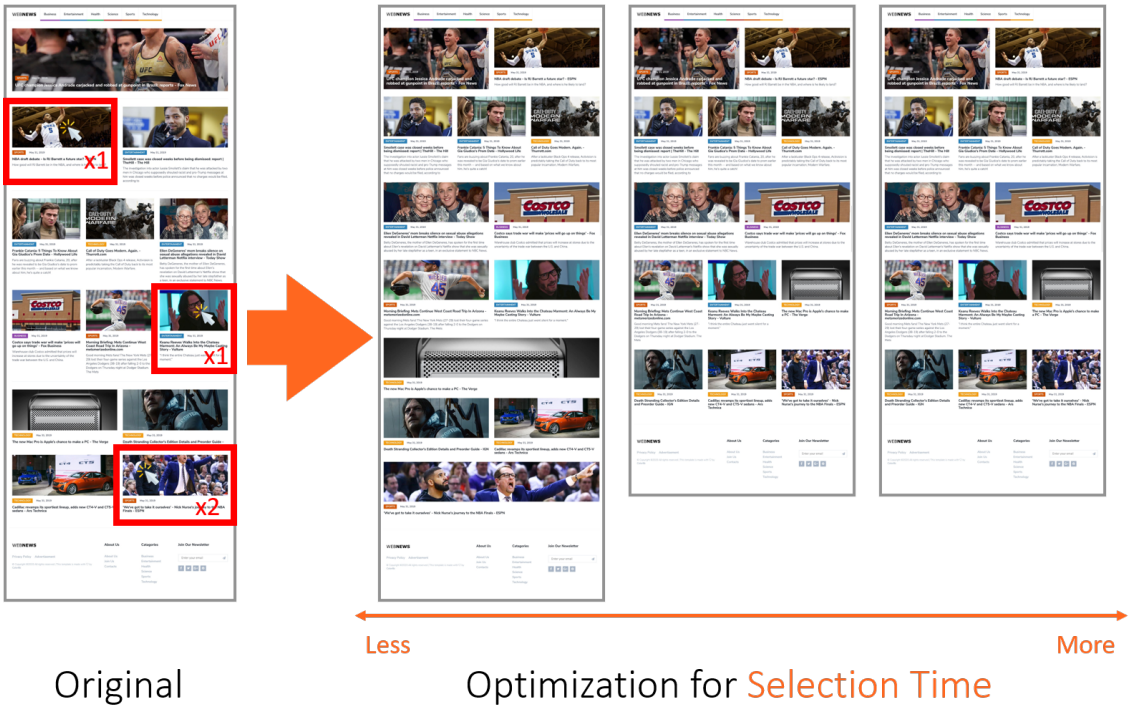


Figure 26: Evaluation result for the selection time objective.

In Figure 26, the first interface on the left is the original input interface. The number and click icon on an element in the interface represent the number of clicks on that element. The more clicks, the more interested the user is in the element. Then the importance of this element is higher. The three interfaces on the right are the optimized interfaces with the optimization weights of 0, 0.5, 1.0 from left to right for selection time. It can be seen that C-RWD optimizes the selection time for the three elements that are clicked. Since C-RWD uses Fitts' law to calculate an element's selection time, the selection time of an element is related to its size and position in the interface. To optimize the selection time, C-RWD will reduce the distance from these elements to the upper left corner of the interface and increase the size of them according to the number of clicks. Simultaneously, as the optimization weight of the

selection time objective increases, C-RWD will increase the intensity of optimization.

The second objective is the visual saliency of the elements in the interface. Similarly, the selection time objective value is set to 0, and the layout similarity objective value is set to 1.0 to eliminate the impact of these two objectives. The optimization weights of the visual saliency objective will be set to 0, 0.5, and 1.0 to gradually increase the optimization weights. The optimization effect of C-RWD on the interface is shown in Figure 27.

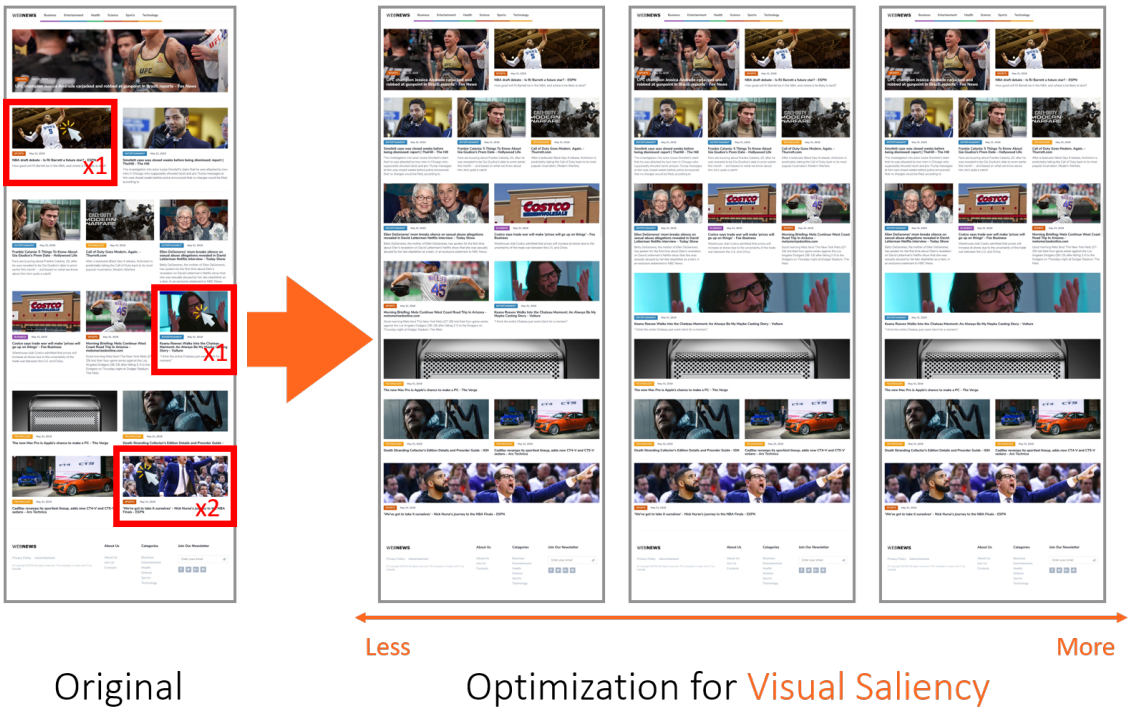


Figure 27: Evaluation result for the visual saliency objective.

Similarly, the left interface in Figure 27 is the original input interface. The number of an element on the interface represents the number of clicks. The three interfaces on the right represent the optimization results when the optimization weight of visual saliency is 0, 0.5, 1.0, respectively. It can be seen that as the optimization weight increases, the sizes of the three clicked elements are adjusted, and their saliency in the interface increases significantly. At the same time, the whole layout keep in the original elements order.

The third objective is the similarity of the interface relative to the original interface. When the other two objectives' optimization weights are 0, the similarity objective's optimization weights are set to 0, 0.5, and 1.0 to observe the difference. The optimized interface effect is shown in Figure 28. The left side of Figure 28 is the original input

interface. The number of clicks on the element is marked on the interface. It can be seen that when similarity is 0, the optimized interface will sort the elements in the interface according to the importance of each element, which could be calculated from the number of clicks. When similarity is 1, the optimized interface will retain the order of elements in the original interface. When the similarity is 0.5, the elements' order is determined by its importance value and original order together according to Equation 2. For example, the "car" element has been promoted from the bottom of the interface to the interface's top due to its high importance. However, although its importance is higher than the "basketball player" element, it is still behind the "basketball player" element to partly follow the original interface's order.

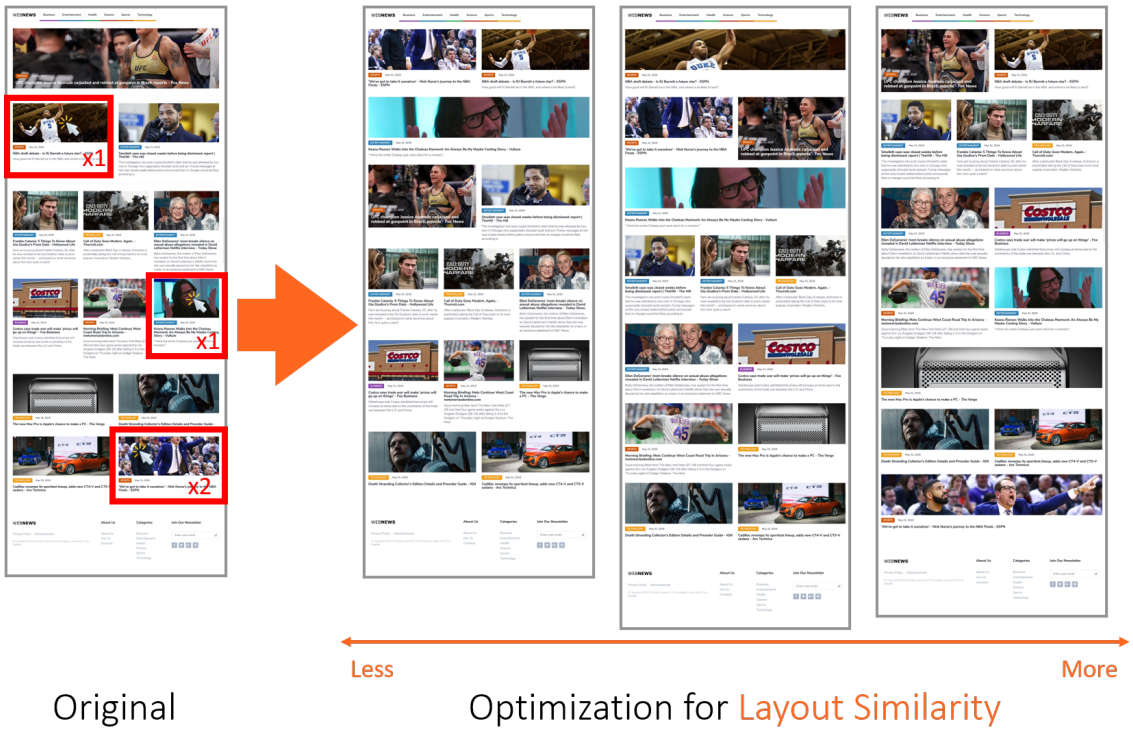


Figure 28: Evaluation result for the similarity objective.

5.1.2 Different Viewport Width

This subsection will apply the same optimization objectives to different viewports width design tasks to generate optimized interfaces.

When the input interface is a fully static interface with a width of 320 pixels, C-RWD will generate a responsive interface with different layout at each breakpoint. In this process, the optimization objectives in the design task are all set to 1. In other words, the layout generator will take care of the three objectives of selection time, visual

saliency, and similarity at the same time. The generated interface is shown in Figure 29.

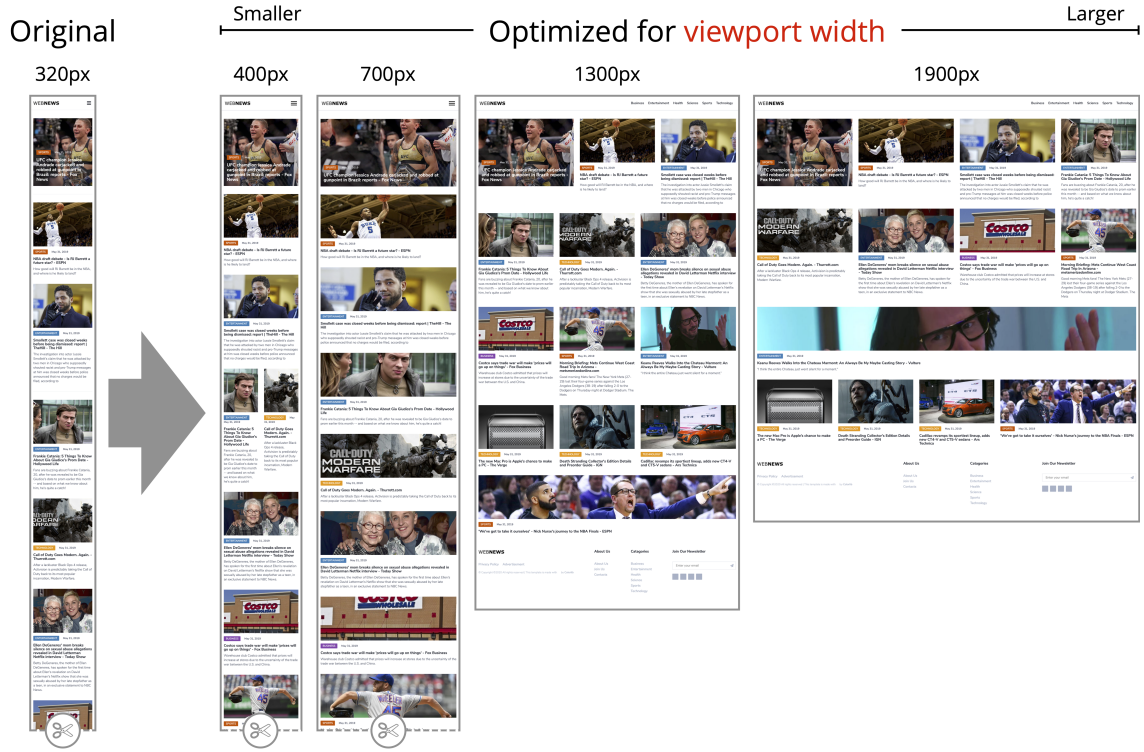


Figure 29: Evaluation result for layouts in different viewport width with static mobile input interface (By Markku Laine).

In Figure 29, the interface on the left is the input 320 pixels wide static mobile interface. The interface on the right is the optimized interface under different viewports width. The optimized result of 400 pixels width represents the interface on mobile devices. The 700 pixels wide optimization result is the interface on the tablet device. The optimized result of 1300 pixels wide is the interface on the laptop device. Finally, the optimized result of 1900 pixels wide represents the interface on the desktop device. In this figure, the similarity objective is set to 1, so the elements' order is not adjusted. In order to facilitate the comparison of optimized effects, the elements in the interface are not clicked. It can be clearly seen that C-RWD processes a single static interface on the mobile device and generates a responsive interface with multiple breakpoints.

When the input interface is fully responsive, C-RWD will regenerate the responsive layout optimized based on the user's interaction history for this interface. The comparison between the newly generated interface and the original interface is shown in Figure 30. The devices in the picture are desktop (1600px), laptop (1280px), tablet

(768px) and mobile device (320px). The objectives used by the C-RWD optimized design task are "selection time: 1.0", "visual saliency: 1.0", and "similarity:0". The clicked element and the number of clicks in the input interface are "car element: 2", "basketball player element:1", and "baseball player: 1" respectively. A fully responsive desktop interface serves as the original input interface on the left side of Figure 30. On the right is the optimized, responsive interfaces regenerated after optimization. As can be seen from Figure 30, the position and size of the elements in the generated interface have been adjusted to ensure that important elements have a reduced selection time and better visual saliency. The white space on the left and right sides of the interface on the desktop device is also eliminated.

Figure 30: Evaluation result for layouts in different viewport width with responsive input interface (By Markku Laine).

5.1.3 Usability Improvement

This subsection will demonstrate and analyze the improvement of the usability of the interface by C-RWD. In C-RWD generating responsive interface, some features and functions can help the generated interface have better usability. For example, C-RWD uses the `isIdealCandidate` attribute to make the layout generator preferentially use elements with optimal line length. Besides, the automatic cropping of image elements by C-RWD will also make image elements have good usability. A specific example is shown in Figure 31.

In this example, a static mobile interface with a width of 320 pixels will be used as the input interface, as shown in the left interface of Figure 31. Simultaneously, to clearly show how C-RWD improves the usability of the generated responsive interface, no element in the input interface is clicked. In other words, the importance value of all target elements is 0. Besides, the selection time and visual saliency objectives in the design task are set to 0 while the similarity objective is set to 1.0. In this

Figure 31: Comparison of usability improvements: (a) original layout, (b) generated RWD layout without usability improvement, and (c) generated RWD layout with usability improvement.

way, the interface output by C-RWD will not be affected by the optimization of the layout generator. To compare usability improvement in the responsive interface generated by C-RWD, a set of comparison and experimental interfaces were generated. First, C-RWD disables usability improvement related functions to generate a basic responsive interface as a control interface, as shown in the middle interface of Figure 31. Then, correspondingly, C-RWD enables the usability improvement function to generate an improved responsive interface as the experimental group, as shown in the right interface of Figure 31. It can be clearly seen that the width of the elements in the interface of the control group is obviously too wide, resulting in poor readability and visual presentation. Simultaneously, the picture elements in the control group interface were not automatically cropped and adjusted, which failed to display important information in the pictures, such as faces and signs. The experimental group interface elements have better shapes and sizes, which improves the text reading experience. Moreover, the picture elements in the experimental group interface have a more reasonable display area after being automatically cropped and adjusted.

In short, from this example, we can see that C-RWD improves the usability of the interface in the process of generating a responsive interface, making the interface more like being designed by a real designer.

5.1.4 Overall Result

This subsection will show the overall interface generated by C-RWD.

For the example of the Webnews website, when the input interface is a static mobile interface with a width of 320 pixels, the process of C-RWD generating an optimized, responsive interface for the interface based on user interaction events is shown in Figure 32. It can be seen that over time, users clicked on different elements. C-RWD will optimize the input interface and generate a responsive interface based on these click events and combined with preset objectives. The optimization objectives used by C-RWD in Figure 32 are "selection time: 1.0, saliency: 1.0, similarity: 0.25". The C-RWD server will generate an updated, optimized layout for this user at regular intervals, based on the user's recent interaction events.

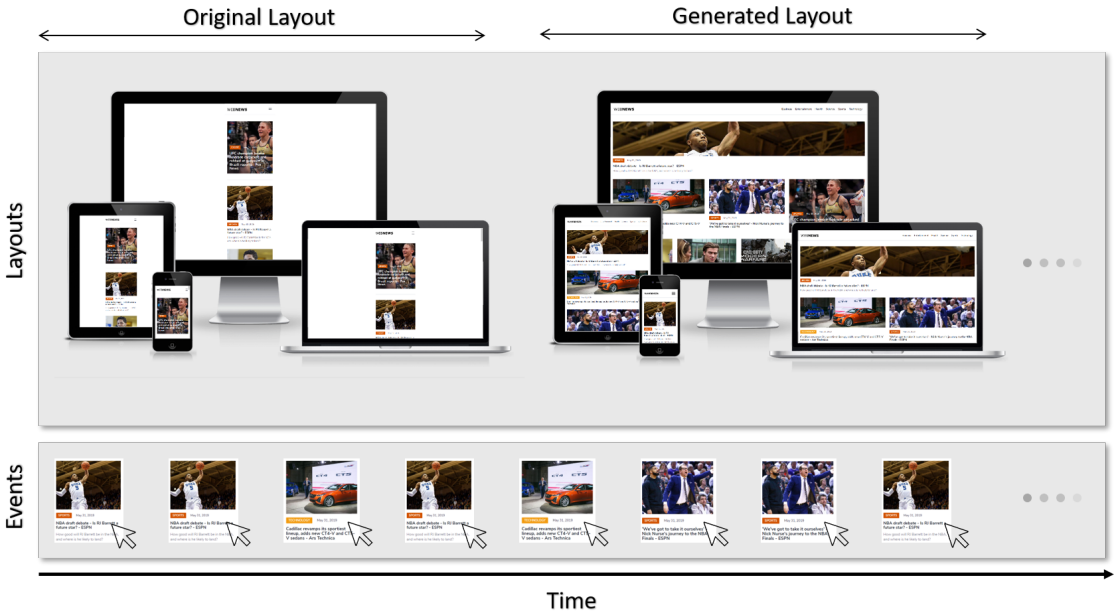


Figure 32: Overall output result of C-RWD.

5.2 RWD Techniques Comparison

The previous subsection demonstrated and evaluated C-RWD's output interface for different input interfaces and RWD design tasks. The results of the evaluation show

that C-RWD has solved the problems raised in research questions. This subsection will compare C-RWD with other common RWD technologies to understand the advantages and existing shortcomings of C-RWD. This will also play a guiding role in analyzing the limits of C-RWD in the subsequent sections.

As mentioned in Section 3, the technologies for implementing responsive web design and development include CSS framework, JS framework, visual web builders. In this subsection, each type of technology will be selected one or two specific technologies to compare with C-RWD. Their specific conditions have been introduced in Section 3, so this subsection will only compare them. For visual web builders, teleportHQ and SquareSpace are selected as representatives. Bootstrap and Masonry are chosen as representatives of the CSS framework and JS framework, respectively. The comparison is carried out from the following aspects. Automated RWD refers to the ability of the tool to generate RWD designs and codes automatically. Personalized breakpoints refer to whether the tool supports breakpoints adjustment to achieve a personalized, responsive interface. Layout optimization refers to whether the tool can optimize the interface for a certain design objective. The code generation indicates whether the tool can automatically generate code for the interface. Automated updating indicates whether the tool supports automatic maintenance and update of the interface. Complex layout support refers to whether the tool supports complex layouts such as nested containers layout. Manual workload refers to the amount of manual work that still needs to be invested when using the tool to create a responsive interface. Finally, learning cost refers to the learning cost that needs to be invested when using the tool to create a responsive interface. The results of the comparison are shown in Table 7.

It can be seen from Table 7 that in terms of "automated RWD", only C-RWD and Masonry support this feature. Other technologies require website designers and developers to create responsive pages manually. Masonry only supports container components for the automatic responsive layout design of the internal waterfall elements but does not support the entire interface's automatic responsive design. Squarespace supports automatically generate responsive interface code, but it needs users to design the responsive interface on a predefined template manually.

In terms of "personalized breakpoints", only C-RWD supports this feature. Other technologies use preset breakpoints (teleportHQ, Squarespace, Bootstrap) or do not support breakpoints settings (Masonry). Although Bootstrap supports manually modify the predefined breakpoints, it is usually technically challenging for users to

Table 7: Comparison among different RWD techniques.

Techniques / Features	C-RWD	teleportHQ	Squarespace	Bootstrap	Masonry
Automated RWD	Yes	No	No	No	Yes
Personalized breakpoints	Yes	No	No	Yes	No
Layout optimization	Yes	No	No	No	No
Code generation	No	Yes	Yes	No	No
Automated layout updating	Yes	No	No	No	No
Complex layout support	Conditional	Yes	No	Yes	No
Manual work load	Low	Medium	Low	High	Low
Learning cost	Low	Medium	Low	High	Low

customize the framework. In terms of layout optimization, only C-RWD supports this feature. Although teleportHQ supports AI to assist designers in making decisions about elements' location when creating the interface. However, only C-RWD supports objective-based optimization of the entire interface to get an optimal design.

For "code generation", teleportHQ supports converting user-designed wireframe interfaces into web code. Squarespace supports the generation of corresponding codes for web content adjusted by users in the content management process. C-RWD and other technologies do not support this feature for the time being. For C-RWD, the user needs to provide an input layout as the starting interface. In the future, C-RWD can be combined with other visual web builders to help users create a responsive interface from scratch. For "automated layout updating", only C-RWD can optimize

the interface multiple times according to user interaction history events to update the interface layout used by different users. Other technologies do not support this feature and require developers to maintain and adjust the interface manually.

For "complex layout support", C-RWD conditionally supports this feature. When the interface is complicated, the user may need to manually mark the interface's key elements to help the C-RWD identify and process all the key elements. Also, when there are multiple container elements in the interface, C-RWD's layout generator does not support optimizing it temporarily. However, the layout generator can be expanded in the future to support the optimization of the interface with multiple container elements. TeleportHQ and Bootstrap benefit from requiring users to manually create interfaces, hence enabling users to create complex interfaces manually. This, by the way, puts forward higher requirements for users.

As for "manually work load", C-RWD generally only needs one line of code to introduce C-RWD's loader component. Therefore, the user does not need to perform manual work. Squarespace and Masonry also require users to fill in the template with website content according to their rules. However, teleportHQ and Boostrap require users to create interfaces manually, which requires more manual workload.

Finally, in terms of learning cost, C-RWD, Squarespace, and Mansory have plug-and-play features and almost no need for users to learn new knowledge. Moreover, teleportHQ requires users to learn how to use its platform and design a responsive layout according to its rule. Bootstrap, as a CSS framework with rich responsive features such as the Bootstrap grid, requires users to invest much time to learn and master it.

5.3 Limitations

The previous subsection compared C-RWD with other RWD technologies and summarized the advantages and disadvantages of each technology. This subsection will mainly introduce the existing limitations of C-RWD.

In terms of interface optimization, C-RWD has the following limitations. First of all, C-RWD does not currently support optimization for interfaces with multiple container elements. In other words, when there are complex element groups in the input interface, the layout generator currently cannot support the simultaneous optimization of multiple groups of elements. This can be achieved through the expansion of the layout generator. Second, the layout generator preferentially uses

shapes that conform to the optimal CPL. Some elements cannot find the ideal shape in some viewport widths and will use other shapes, which will cause them to have alignment problems with other elements that use ideal shapes. Third, because the layout generator's optimizer has slightly different performance on different platforms, under the same input interface and design task, the layout of the interface generated by C-RWD on different platforms is slightly different.

In terms of responsive interface generation, first, because C-RWD uses the grid layout, C-RWD may not support interface design patterns other than grid design patterns, such as fixed-size layout patterns for some elements. Second, the responsive interface generated by C-RWD still has problems such as the excessive width of some elements in some extreme cases. This requires C-RWD to further process and limit the aspect ratio of interface elements. Third, for the static navigation bar elements of the input interface, C-RWD will lose some of its original style design in the process of generating a responsive navigation bar. Simultaneously, C-RWD does not currently support the responsive conversion of complex navigation bars with multi-level menu elements.

In terms of usability improvement, C-RWD currently selects the shape of elements through a shape set. At the same time, it combines optimal line length and intelligent image cropping to improve usability. Some styles inside the element, such as font size and image position, are not considered due to computing power limitations. Therefore, the optimized elements of C-RWD may still have some subtle problems.

In terms of user testing, C-RWD currently focuses on the realization and evaluation of functions, instead of conducting detailed user testing to verify the interface generated by C-RWD from the user's perspective. Therefore, the current evaluation of C-RWD is only conducted from a technical perspective. In the future, C-RWD needs more tests and experiments from the user's perspective to evaluate the use of C-RWD fully.

5.4 Future Work

The previous subsection discussed some of the limitations of C-RWD. Based on the current C-RWD framework and some existing defects, this subsection looks forward to the aspects that C-RWD can continue to improve in the future.

In terms of C-RWD system configuration, C-RWD currently supports website developers to customize their configuration by providing configuration documents, such as

setting optimization objectives and breakpoints. In the future, in order to facilitate website developers to adjust the configuration of C-RWD, C-RWD will integrate relevant configuration parameters into the existing visual control panel of the LaaS platform. In this way, C-RWD users can conveniently adjust various parameters and options of C-RWD through this control panel interface. Besides, it can be seen from Table 7 that C-RWD does not support the generation of interface codes. Therefore, one of the development directions in the future is to shift from requiring users to provide a starting interface to only requiring target elements. In this way, users can use C-RWD with fewer start codes. C-RWD can also cooperate with other visual web builders to help users quickly create the initial interface required by C-RWD.

In terms of interface optimization, C-RWD needs to adjust the layout generator's optimization algorithm to support the optimization of the interface with multiple containers. The adjusted algorithm's general idea is to optimize the elements in each group of containers from bottom to top. For example, for an interface with three container elements under the body element, the layout generator will first optimize each container's elements. After getting the optimization results, the layout generator then treats these container elements as target elements to optimize and output the final interface. Also, for the alignment problem between non-ideal shape elements and other elements, C-RWD can solve this problem by relaxing constraints and reducing non-ideal shape elements. For the different output results of the layout generator under the same input, this will be solved in the future by using the same kind of layout optimizer's operating environment and fixed operating parameters on the server-side. Finally, since C-RWD can add or adjust optimization objectives in the future to improve the effect of interface optimization, more optimized objectives models can be applied to the optimization algorithm of the layout generator.

In terms of the generation of responsive interfaces, C-RWD will need to use more different layout methods to support different RWD design patterns in the future. This allows C-RWD to meet more different types of RWD needs. For the aspect ratio of the element, C-RWD will filter out inappropriate shapes by adding the aspect ratio detection of the target element in the layout parser component. Excessive changes in the aspect ratio of an element often lead to deformation of the element content and decreased user experience. Finally, for the responsive conversion of navigation bar elements, C-RWD can further refine the original navigation bar styles' analysis process and transplant them to the newly generated navigation bar. In addition, deep learning methods can also be embedded to support the migration of navigation

bar styles.

In terms of usability improvement, C-RWD can use more interface design principles in the future to adjust the internal styles of elements such as font size. Besides, machine learning can also be introduced to help adjust the internal details of elements. This can help the C-RWD generate an interface with better usability.

In terms of system evaluation, C-RWD will need to conduct a series of user tests in the future to evaluate the usability of the generated responsive interface and provide more ideas for improvement from the user's perspective. C-RWD can also invite experts in responsive interface design and development to conduct an expert evaluation of the generated results to find problems.

In short, because the C-RWD framework has good modularity and scalability, C-RWD can easily improve and adjust various components to enhance its functions in the future. LaaS platform can integrate other tools such as visual web builder to work with C-RWD. Other optimization methods, such as machine learning, can also be introduced to assist combinatorial optimization to adjust the interface's details.

6 Related Work

This section will conduct a literature review of the related work of C-RWD. Since C-RWD covers mainly layout parsing, layout optimization, and building responsive interface, this section will introduce relevant research in these three areas and compare them with C-RWD.

6.1 Layout Parsing

Layout parsing is the first step in user interface generation and redesign. In order to restructure the input interface, the page structure of the interface should be parsed to obtain the DOM structure of the interface, target elements, and other information. Layout parsing is usually implemented in two ways. For interfaces with common structures, layout parsing can analyze the interface structure and find target elements through heuristic element label semantic analysis. C-RWD performs heuristic layout parsing by analyzing the text semantics of the element label and combining common framework rules. In addition, [64] also proposed a method to analyze top-down elements of the interface based on heuristic rules. In recent research, [65] used a partitioning algorithm based on clustering [66] to segment the interface after evaluating a variety of different web page segmentation methods to help automatically fix the mobile interface friendliness problem. Layout parsing based on heuristic rules often has high accuracy. However, its disadvantage is that it may require manual annotation of some elements to assist in analyzing the interface with a special structure. Another more versatile layout parsing method is to analyze the interface through artificial intelligence. For example, ReMorph uses the language parsing algorithm in natural language processing, the CYK algorithm, to intelligently group and analyze elements [67] when parsing the page. In addition to NLP algorithms, machine learning is also used for layout parsing. For example, Baluja et al. proposed a machine learning framework to intelligently analyze the interface through decision trees and entropy reduction [68]. Machine learning can also use computer vision methods to make full use of the interface's visual information such as font style, element size, color, etc. to resolve the interface [69]. For example, Xie et al. used the VIPS algorithm [70] to analyze the website interface based on vision [69]. The layout parsing method based on artificial intelligence has better versatility and can make full use of various interface information. However, their disadvantage is that the analytical results are not accurate enough, which affects the automatic redesign of the interface.

6.2 Layout Optimization

Interface layout optimization is an important part of layout generation and redesign. This subsection will introduce the two types of layout optimization, rule-based optimization and model-based optimization. Then, this subsection will introduce different optimization methods for different user groups.

6.2.1 Rule-based Approaches

Rule-based optimization refers to optimizing the layout of the interface through the guideline or heuristics rules of the layout design. It can simply and directly optimize the layout of the interface to improve the usability of the layout. For example, [71] integrates solutions to problems that may occur in web page layout on large screen devices into a web page template to optimize the interface layout. For different types of elements in the interface, [72] proposed Xadaptor to optimize their usability in interfaces of different widths by providing their adaptation rule base for different elements. The disadvantage of rule-based optimization is that it often only supports specific types of websites or elements. Therefore, its scalability is poor. Even if some rule-based optimization frameworks are extensive, such as Xadaptor [72], it needs to manually add and maintain its rule base for different types of elements. Besides, rule-based optimization cannot optimize the interface for a specific design objective.

6.2.2 Model-based Approaches

Model-based optimization refers to abstracting the interface's layout through various models and optimizing the interface by optimizing the objective function based on the model. Model-based optimization usually includes combinatorial optimization methods based on white boxes and machine learning methods based on black boxes.

Combinatorial optimization has many applications in the optimization of interface layout. [3] covers different aspects and related knowledge on combinatorial optimization of GUI design, such as design task definition, optimization modeling, optimization formulation for different layout problems, and practice deployment. The advantage of combinatorial optimization is that it can be optimized for a specific design objective, and the optimal solution can be guaranteed in a limited time.

Combinatorial optimization can customize the interface for different users through user traces [73, 74, 4, 6]. For example, SUPPLE uses user traces to provide different users with different interface rendering methods based on the usage pattern model

[73]. Familiarisation automatically generates personalized interface design for users by analyzing the user's historical visited interface and visual learning model [74, 4]. LaaS analyzes user preferences by collecting and analyzing user interaction history, such as click events, and uses predictive models such as Fitts' law to optimize the user's element selection and click time. C-RWD uses a personalized optimization method similar to LaaS to produce personalized, responsive designs for users. Combinatorial optimization can also be used to help designers interactively design interface layouts. For example, [5] proposed GRIDS to provide interface designers with interactive layout design suggestions. For distributed interfaces, combinatorial optimization can also be used to perform foraging-based optimization [75] and optimization of multi-user cooperation [76] for interfaces under different usage contexts. For example, [75] uses integer linear programming (ILP) to generate candidate layouts that meet the design standards and constraints of the display device to improve information acquisition efficiency. For the interface used by multi-user cooperation, AdaM dynamically adjusts the layout of different user roles through integer optimization to optimize the user experience [76]. Besides, combinatorial optimization also has many applications in document formatting [77], such as the layout optimization of tables in documents under different widths [78].

Machine learning also has many applications in interface layout optimization. Machine learning can be used to automatically generate interfaces for mobile devices to improve user browsing efficiency [69], interactively help designers to design ideation [29], interface skeleton code generation based on UI image design [30], and perform UI automatic testing [31], etc. For example, Xie et al. proposed a machine learning framework based on the block importance model to automatically label importance values for the interface elements and redesign the generated mobile interface according to its importance [69]. C-RWD also uses an importance model based method to optimize the interface. However, the difference is that C-RWD uses user tracking methods to record user interaction data and generate personalized element importance values for users. While [69] uses the support vector machine method to generate importance values of interface elements through the extraction and analysis of element features. However, machine learning methods often require a large amount of data to support the training of the model, while combinatorial optimization usually requires only a small amount of user data to optimize the interface.

6.2.3 Personalized Optimization

In user interface optimization, the optimization process could be personalized for different user group levels using personalized optimization parameters. Some optimization methods are optimized based on the entire user population [5, 69, 75, 79]. They optimize the interface from the interface's overall aesthetics and element features such as content length, font color, and interface display space limitations, without considering the differences between interface users. For example, Raneburger et al. proposed a multi-strategy GUI tailoring method based on integer programming to optimize the interface for display devices of different sizes [79]. This optimization method considers different heuristic optimization strategies without considering user preferences.

Another GUI optimization methods uses ability-based optimization for different user groups [80] or take cultural background [81] into the optimization considerations. In other words, they perform specific optimizations for different subsets of the entire user population. The optimized target user group can be determined according to the user's age [82], motor and visual ability [83, 84], cultural background [81], etc. For example, [82] proposes a text input interface optimized for the elderly with finger tremor to improve the text input speed of the user group on the touch keyboard. Besides, the division of user groups can also group users based on user feedback. For example, [85] proposed a community-based automatic adaptation system for interface content.

The two types of interface optimization mentioned above are optimized for user groups. Besides, the user interface can also be personalized for individual users [4, 74, 73, 86, 87, 88, 6, 89]. C-RWD personalizes the user interface to improve user experience. In this way, the interface can be adjusted according to different preferences of users. However, it requires a certain level of collection of user information.

6.3 Building Responsive Interface

In recent years, the creation of responsive interfaces has attracted the interest of many researchers and website developers [33, 90, 91]. [33] introduces the reasons for using a responsive interface, development techniques, benefits, and existing problems. [90, 91] conducted a detailed literature study on the methods used in responsive development. This subsection explains the related research on building responsive interfaces from two aspects: interactive RWD and automated RWD.

6.3.1 Interactive RWD

Interactive RWD makes interface designer in the design loop. For example, [92] proposed Expresso, which can help designers directly manipulate interface elements in the system without programming. Expresso defines a concept called "Keyframes" for the design of responsive interfaces. Designers can design the keyframes of the interface in the system. Expresso will automatically generate responsive transition interfaces for these keyframe interfaces. In addition, [93] proposed DECOR to help designers interactively create RWD. Given the initial interface and constraints, DECOR can interactively generate design candidates for different interface widths for designers to choose and use. Interactive RWD can reduce the burden of responsive design on designers. However, it requires the designer to participate and provide the starting interface or design the key interface.

6.3.2 Automated RWD

Compared with interactive RWD, Automated RWD further reduces the workload of RWD creation. The RWD method proposed by C-RWD is a method that completely automatically generates and optimizes a responsive interface. However, according to the related literature search, there is no related research that can automatically generate a responsive interface while optimizing the interface.

Automated RWD has multiple implementation methods. A common way is that the automated RWD system provides a rule-based model language for responsive interface development to help the dynamic generation and adaptation of the interface [94, 95, 96, 97]. For example, [97] proposed SuperSQL, a responsive interface modeling language based on the SQL database. SuperSQL can automatically generate a responsive interface through related rules stored in the database. The disadvantage of these rule-based modeling languages is that they usually have poor scalability. Users still need to create corresponding model rules for different interfaces manually. Compared with these methods, C-RWD only needs one input interface of the interface to automatically generate a responsive interface without any manual setting based on rules.

Another common automated RWD method is to parse the interface and generate a responsive interface automatically. For example, [67] proposes a method to generate a mobile responsive interface for the input interface automatically. It can generate a mobile interface by analyzing the grouping relationship and sequence of elements in the original interface. However, this method does not support interface optimization.

Compared with this method, C-RWD can not only automatically generate RWD, but also optimize the responsive interface based on user interaction history. In addition, [98] proposes a method that can automatically generate a responsive interface for mobile devices in landscape and portrait modes. In contrast, C-RWD not only supports the generation of interfaces in landscape and portrait modes for mobile devices but also supports the generation of desktop interfaces.

The template-based method can also be used to automatically or semi-automatically help generate a responsive interface [71, 99]. For example, [71] proposes a method based on a responsive template to help generate a responsive interface and adjust the text size, layout, and media content of the interface. This template-based method has poor versatility, and it is often only used to generate specific types of responsive interfaces.

In addition to the above automated RWD methods, because the responsive interface has different interface layouts under different widths, some usability problems are often difficult to detect by developers. Some responsive interface automatic tests [100, 101, 102] and problem fixes [65] can help developers improve the quality of responsive interfaces. For example, Mahajan et al. proposed an automatic problem detection and repair method for the mobile interface to improve the readability of the mobile interface [65].

7 Conclusion

As a service based on the LaaS platform, C-RWD supports converting the input interface into a fully responsive interface while optimizing it based on multiple objectives. It makes the LaaS platform evolved from only supporting the generation of a static optimized interface to supporting the generation of a fully responsive optimized interface. At the same time, in the process of generating the responsive interface, some problems affecting the usability of the interface are also well solved by C-RWD. As shown in Section 4 and Section 5, C-RWD solves all three research questions very well. For RQ1, C-RWD's DTG and event logger components get the user device information and automatically generate personalized breakpoints for different design tasks based on user's history visits. For RQ2, given input layout and design task, C-RWD's layout parser and layout generator components automatically optimize interface layouts in different width based on three design objectives (selection time, visual saliency and similarity) and shape set constraints. For RQ3, the RWD converter and RWD adapter components of C-RWD convert and adapt the optimized layouts into one fully responsive layout and improve the generated interface's usability by features like intelligently cropping the image element. Finally, by solving these three RQs, the final research question is also solved, and the final generated interface is evaluated in Section 5.

C-RWD's theoretical contribution lies in that it proposes a new framework to generate and optimize responsive interfaces. Before this, LaaS used combinatorial optimization to optimize the interface and generate a static interface. The innovation of C-RWD is that C-RWD takes the optimization results of the input interface under different widths as the interface of the responsive interface at different breakpoints. At the same time, C-RWD adjusted the interface optimization process to the characteristics of the responsive interface. After integrating and improving the interface at these optimized breakpoints, C-RWD finally outputs a responsive interface. Secondly, C-RWD proposes a new data-driven design method for responsive interface design. In other words, C-RWD collects and analyzes the user's interaction history through event loggers and personalizes the layout of the interface based on these data. This allows the optimized interface to be personalized for individual users. Simultaneously, this optimization process is not a one-time, but an iterative update and optimization of the interface based on the user's use over time. This provides a new way of thinking for responsive interface design and generation.

The contribution of C-RWD in practical applications lies in the following aspects.

First, for web page designers and developers, C-RWD can greatly reduce the time they need to spend in creating responsive interfaces. They only need to design an initial interface of any width and complete the relevant configuration of the C-RWD. They can then use C-RWD to generate a responsive interface that fits devices of various sizes and automatically optimize it for different users. Second, C-RWD can personalize and optimize the interface they use according to their usage habits for web users. In this way, users can more quickly notice and select the interface content they are interested in. When the website has much different content (for example, a news website), this may greatly improve the user experience.

C-RWD still has some shortcomings, such as not enough RWD design patterns supported and limited example websites evaluated currently. However, thanks to C-RWD's modular design and the LaaS platform's scalability, these limitations can be adjusted and added in the future according to actual needs like supporting more kinds of websites. In other words, C-RWD provides a framework and verifies the feasibility of it. C-RWD can be used as a reference framework for researchers interested in the research topic of automatic generation and optimization of responsive interfaces. Researchers can add more technologies such as machine learning to this framework to enrich the framework's content or explore other possible computational responsive web design methods. For example, researchers can adjust the optimization objectives of C-RWD according to their own research goals to explore the impact of different optimization objectives on the generated responsive interface. In short, C-RWD is not only for the above-mentioned purposes. It can be used as a development framework to help the responsive interface research community explore more research questions.

References

[1] GMDT Forecast. Cisco visual networking index: global mobile data traffic forecast update, 2017–2022. 2019.

[2] Statcounter. Screen resolution stats worldwide. https://gs.statcounter.com/screen-resolution-stats, accessed 2020-11-15.

[3] Antti Oulasvirta, Niraj Ramesh Dayama, Morteza Shiripour, Maximilian John, and Andreas Karrenbauer. Combinatorial optimization of graphical user interface designs. *IEEE Proceedings*, 2020.

[4] Kashyap Todi, Jussi Jokinen, Kris Luyten, and Antti Oulasvirta. Familiarisation: Restructuring layouts with visual learning models. In *23rd International Conference on Intelligent User Interfaces*, pages 547–558, 2018.

[5] Niraj Dayama, Kashyap Todi, Taru Saarelainen, and Antti Oulasvirta. Grids: Interactive layout design with integer programming. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. ACM, 2020.

[6] Markku Laine, Ai Nakajima, Niraj Dayama, and Antti Oulasvirta. Layout as a Service (LaaS): A service platform for self-optimizing web layouts. In *Proc. ICWE'20*, 2020.

[7] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, pages 75–105, 2004.

[8] W3C. HTML5 differences from HTML4. https://www.w3.org/TR/html5-diff/, accessed 2020-11-15.

[9] David Flanagan and Gregor M Novak. Java-Script: The Definitive Guide, 1998.

[10] Håkon Wium Lie. Cascading HTML style sheets-a proposal. *World Wide Web Consortium (W3C)*, 1994.

[11] Håkon Wium Lie and Bert Bos. Cascading style sheets, level 1. *Recommendation RECCSS1, World Wide Web Consortium*, 1996.

[12] Mozilla.org. Javascript introduction in MDN web docs. https://developer.mozilla.org/en-US/docs/Web/JavaScript , accessed 2020-11-15.

[13] Charles Severance. Javascript: Designing a language in 10 days. *Computer*, 45(2):7–8, 2012.

[14] G Paolini. Netscape and Sun announce JavaScript, the open cross-platform object scripting language for enterprise networks and the internet. *Press Release]. Sun Microsystems, Inc*, 1995.

[15] Smashing Magazine. *The Smashing Book# 1*. Smashing Media GmbH, 2011.

[16] W3schools.com. Browser display statistics. http://www-db.deis.unibo.it/courses/TW/DOCS/w3schools/browsers/browsers_display.asp.html/, accessed 2020-11-15.

[17] W3C. Cascading style sheets level 2 revision 1 (CSS 2.1) specification. https://www.w3.org/TR/CSS2/, accessed 2020-11-15.

[18] Ethan Marcotte. Fluid grids. https://alistapart.com/article/fluidgrids/, accessed 2020-11-15.

[19] Mozilla.org. Font size. https://developer.mozilla.org/en-US/docs/Web/CSS/font-size, accessed 2020-11-15.

[20] W3C. CSS flexible box layout module level 1, Nov 2018. https://www.w3.org/TR/2018/CR-css-flexbox-1-20181119/, accessed 2020-11-15.

[21] Caniuse.com. Caniuse website search result for flexbox. https://caniuse.com/#search=flexbox, accessed 2020-11-15.

[22] W3C. CSS grid layout module level 1, Dec 2017. https://www.w3.org/TR/css-grid-1/, accessed 2020-11-15.

[23] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[24] John Williamson, Antti Oulasvirta, Otmar Hilliges, and Per Ola Kristensson. Computational interaction: Theory and practice. In *Proceedings of the 2018 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, 2018.

[25] Franz Rothlauf. *Design of modern heuristics: principles and application*. Springer Science & Business Media, 2011.

[26] Paul M Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of experimental psychology*, 47(6):381, 1954.

[27] Simon Lok and Steven Feiner. A survey of automated layout techniques for information presentations. *Proceedings of SmartGraphics*, 2001:61–68, 2001.

[28] Gurobi.com. Gurobi. https://www.gurobi.com/, accessed 2020-11-15.

[29] Janin Koch, Andrés Lucero, Lena Hegemann, and Antti Oulasvirta. May AI? Design ideation with cooperative contextual bandits. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2019.

[30] Chunyang Chen, Ting Su, Guozhu Meng, Zhenchang Xing, and Yang Liu. From UI design image to GUI skeleton: a neural machine translator to bootstrap mobile GUI implementation. In *Proceedings of the 40th International Conference on Software Engineering*, pages 665–676, 2018.

[31] Gang Hu, Linjie Zhu, and Junfeng Yang. AppFlow: using machine learning to synthesize robust, reusable UI tests. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 269–282, 2018.

[32] Ethan Marcotte. Responsive web design, 2010. https://alistapart.com/article/responsive-web-design/, accessed 2020-11-15.

[33] Sanja Mohorovičić. Implementing responsive web design for enhanced web presence. In *2013 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1206–1210. IEEE, 2013.

[34] Statcounter. Platform market share stats worldwide. https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet, accessed 2020-11-15.

[35] W3C. The CSS 2 editor's draft, 7 july 2020. https://drafts.csswg.org/css2/#propdef-float, accessed 2020-11-15.

[36] Apple. Configuring the viewport. https://developer.apple.com/library/archive/documentation/AppleApplications/Reference/SafariWebContent/UsingtheViewport/UsingtheViewport.html, accessed 2020-11-15.

[37] Peter-Paul Koch. A tale of two viewports, 2011. https://www.quirksmode.org/mobile/viewports.html, accessed 2020-11-15.

[38] Peter-Paul Koch. Meta viewport. https://www.quirksmode.org/mobile/metaviewport/, accessed 2020-11-15.

[39] W3C. Media queries W3C recommendation 19 june 2012. https://www.w3.org/TR/2012/REC-css3-mediaqueries-20120619/, accessed 2020-11-15.

[40] Mozilla.org. Responsive images. https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Responsive_images, accessed 2020-11-15.

[41] W3C. CSS box sizing module level 4 editor's draft, 4 august 2020. https://drafts.csswg.org/css-sizing-4/#aspect-ratio, accessed 2020-11-15.

[42] Chris Coyier. Aspect ratio boxes. https://css-tricks.com/aspect-ratio-boxes/, accessed 2020-11-15.

[43] W3C. CSS display module level 3 editor's draft, 6 june 2020. https://drafts.csswg.org/css-display/#block-formatting-context, accessed 2020-11-15.

[44] W3C. Web Content Accessibility Guides (WCAG) 2.1 W3C Recommendation 05 June 2018. https://www.w3.org/TR/WCAG21/, accessed 2020-11-15.

[45] Aliaksei Miniukovich, Michele Scaltritti, Simone Sulpizio, and Antonella De Angeli. Guideline-based evaluation of web readability. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2019.

[46] W3C. CSS values and units module level 3 editor's draft, 4 august 2020. https://drafts.csswg.org/css-values-3/#ch, accessed 2020-11-15.

[47] Meyerweb.com. What is the CSS ch unit? https://meyerweb.com/eric/thoughts/2018/06/28/what-is-the-css-ch-unit/, accessed 2020-11-15.

[48] Bootstrap. https://getbootstrap.com/, accessed 2020-11-15.

[49] Zurb foundation framework. https://get.foundation/index.html, accessed 2020-11-15.

[50] Masonry. https://masonry.desandro.com/, accessed 2020-11-15.

[51] Packery. https://packery.metafizzy.co/, accessed 2020-11-15.

[52] Isotope. https://isotope.metafizzy.co/, accessed 2020-11-15.

[53] Fusioncharts.com. Comparing jquery grid plugins. https://www.fusioncharts.com/blog/comparing-jquery-grid-plugins-masonry-vs-isotope-vs-packery-vs-gridster-vs-shapeshift-vs-shuffle-js/, accessed 2020-11-15.

[54] Squarespace. https://www.squarespace.com/, accessed 2020-11-15.

[55] Wix. https://www.wix.com/, accessed 2020-11-15.

[56] Wordpress. https://wordpress.org/ , accessed 2020-11-15.

[57] TeleportHQ. https://teleporthq.io/, accessed 2020-11-15.

[58] Cloudinary. https://cloudinary.com/, accessed 2020-11-15.

[59] ImageKit.io. https://imagekit.io/, accessed 2020-11-15.

[60] Ian Sommerville. Software engineering. 10th. In *Book Software Engineering. 10th, Series Software Engineering*. Addison-Wesley, 2015.

[61] Ruth Rosenholtz, Yuanzhen Li, Jonathan Mansfield, and Zhenlan Jin. Feature congestion: A measure of display clutter. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '05, page 761–770, New York, NY, USA, 2005. Association for Computing Machinery.

[62] Gilles Bailly, Antti Oulasvirta, Timo Kötzing, and Sabrina Hoppe. Menuoptimizer: Interactive optimization of menu systems. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, page 331–342, New York, NY, USA, 2013. Association for Computing Machinery.

[63] Robert Bringhurst. *The elements of typographic style*. Hartley & Marks Vancouver, 2004.

[64] Yu Chen, Wei-Ying Ma, and Hong-Jiang Zhang. Detecting web page structure for adaptive viewing on small form factor devices. In *Proceedings of the 12th international conference on World Wide Web*, pages 225–233, 2003.

[65] Sonai Mahajan, Negarsadat Abolhassani, Phil McMinn, and William GJ Halfond. Automated repair of mobile friendly problems in web pages. In *Proceedings of the 40th International Conference on Software Engineering*, pages 140–150, 2018.

[66] Richard Romero and Adam Berger. Automatic partitioning of web pages using clustering. In *International Conference on Mobile Human-Computer Interaction*, pages 388–393. Springer, 2004.

[67] Gilbert Louis Bernstein and Scott Klemmer. Towards responsive retargeting of existing websites. In *Proceedings of the adjunct publication of the 27th annual ACM symposium on User interface software and technology*, pages 119–120, 2014.

[68] Shumeet Baluja. Browsing on small screens: recasting web-page segmentation into an efficient machine learning framework. In *Proceedings of the 15th international conference on World Wide Web*, pages 33–42, 2006.

[69] Xing Xie, Gengxin Miao, Ruihua Song, Ji-Rong Wen, and Wei-Ying Ma. Efficient browsing of web search results on mobile devices based on block importance model. In *Third IEEE International Conference on Pervasive Computing and Communications*, pages 17–26. IEEE, 2005.

[70] Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. Vips: a vision-based page segmentation algorithm. 2003.

[71] Michael Nebeling, Fabrice Matulic, Lucas Streit, and Moira C Norrie. Adaptive layout template for effective web content presentation in large-screen contexts. In *Proceedings of the 11th ACM symposium on Document engineering*, pages 219–228, 2011.

[72] Jiang He, Tong Gao, Wei Hao, I-Ling Yen, and Farokh Bastani. A flexible content adaptation system using a rule-based approach. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):127–140, 2006.

[73] Krzysztof Gajos and Daniel S Weld. SUPPLE: automatically generating user interfaces. In *Proceedings of the 9th international conference on Intelligent user interfaces*, pages 93–100, 2004.

[74] Kashyap Todi, Jussi Jokinen, Kris Luyten, and Antti Oulasvirta. Individualising graphical layouts with predictive visual search models. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 10(1):1–24, 2019.

[75] Maria L Montoya Freire, Dominic Potts, Niraj Ramesh Dayama, Antti Oulasvirta, and Mario Di Francesco. Foraging-based optimization of pervasive displays. *Pervasive and Mobile Computing*, 55:45–58, 2019.

[76] Seonwook Park, Christoph Gebhardt, Roman Rädle, Anna Maria Feit, Hana Vrzakova, Niraj Ramesh Dayama, Hui-Shyong Yeo, Clemens N Klokmose, Aaron Quigley, Antti Oulasvirta, et al. Adam: Adapting multi-user interfaces for collaborative environments in real-time. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2018.

[77] Nathan Hurst, Wilmot Li, and Kim Marriott. Review of automatic document formatting. In *Proceedings of the 9th ACM symposium on Document engineering*, pages 99–108, 2009.

[78] Mihai Bilauca and Patrick Healy. A new model for automated table layout. In *Proceedings of the 10th ACM symposium on Document engineering*, pages 169–176, 2010.

[79] David Raneburger, Hermann Kaindl, and Roman Popp. Strategies for automated GUI tailoring for multiple devices. In *2015 48th Hawaii International Conference on System Sciences*, pages 507–516. IEEE, 2015.

[80] Jacob O Wobbrock, Shaun K Kane, Krzysztof Z Gajos, Susumu Harada, and Jon Froehlich. Ability-based design: Concept, principles and examples. *ACM Transactions on Accessible Computing (TACCESS)*, 3(3):1–27, 2011.

[81] Andruid Kerne, William A Hamilton, and Zachary O Toups. Culturally based design: embodying trans-surface interaction in rummy. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 509–518, 2012.

[82] Sayan Sarcar, Jussi Jokinen, Antti Oulasvirta, Xiangshi Ren, Chaklam Silpasuwanchai, and Zhenxin Wang. Ability-based optimization: Designing smartphone text entry interface for older adults. In *IFIP Conference on Human-Computer Interaction*, pages 326–331. Springer, 2017.

[83] Krzysztof Z Gajos, Jacob O Wobbrock, and Daniel S Weld. Automatically generating user interfaces adapted to users' motor and vision capabilities. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 231–240, 2007.

[84] Thomas Rathfux, Jasmin Thöner, Hermann Kaindl, and Roman Popp. Combining design-time generation of web-pages with responsive design for improving low-vision accessibility. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 1–7, 2018.

[85] Iqbal Mohomed, Alvin Chin, Jim Chengming Cai, and Eyal de Lara. Community-driven adaptation: Automatic content adaptation in pervasive environments. In *Sixth IEEE Workshop on Mobile Computing Systems and Applications*, pages 124–133. IEEE, 2004.

[86] Krzysztof Z Gajos, Jing Jing Long, and Daniel S Weld. Automatically generating custom user interfaces for users with physical disabilities. In *Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility*, pages 243–244, 2006.

[87] Jeffrey Nichols and Tessa Lau. Mobilization by demonstration: using traces to re-author existing web sites. In *Proceedings of the 13th international conference on Intelligent user interfaces*, pages 149–158, 2008.

[88] Jeffrey Nichols, Brad A Myers, and Brandon Rothrock. Uniform: automatically generating consistent remote control user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 611–620, 2006.

[89] Michael Hinz, Zoltán Fiala, and Frank Wehner. Personalization-based optimization of web interfaces for mobile devices. In *International Conference on Mobile Human-Computer Interaction*, pages 204–215. Springer, 2004.

[90] Nazish Yousaf, Aleena Arshad, Muhammad Nouman, and Umar Arshad. Towards adaptive and responsive web design: A systematic literature review. *Language*, 1:40, 2018.

[91] Nazish Yousaf, Wasi Haider Butt, Farooque Azam, and Muhammad Waseem Anwar. A systematic review of adaptive and responsive design approaches for world wide web. In *Future of Information and Communication Conference*, pages 704–717. Springer, 2018.

[92] Rebecca Krosnick, Sang Won Lee, Walter S Laseck, and Steve Onev. Expresso: Building responsive interfaces with keyframes. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 39–47. IEEE, 2018.

[93] Nishant Sinha and Rezwana Karim. Responsive designs in a snap. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 544–554, 2015.

[94] Dirk Roscher, Grzegorz Lehmann, Veit Schwartze, Marco Blumendorf, and Sahin Albayrak. Dynamic distribution and layouting of model-based user interfaces in smart environments. In *Model-Driven Development of Advanced User Interfaces*, pages 171–197. Springer, 2011.

[95] Enes Yigitbas, Hagen Stahl, Stefan Sauer, and Gregor Engels. Self-adaptive UIs: Integrated model-driven development of UIs and their adaptations. In Anthony Anjorin and Huáscar Espinoza, editors, *Modelling Foundations and Applications*, pages 126–141, Cham, 2017. Springer International Publishing.

[96] Enes Yigitbas. *Model-Driven Engineering of Self-Adaptive User Interfaces*. PhD thesis, Paderborn University, 2019.

[97] Ryosuke Koshijima, Kento Goto, and Motomichi Toyama. Generating responsive web pages using SuperSQL. In *Proceedings of the 18th International Conference on Information Integration and Web-Based Applications and Services*, iiWAS '16, page 231–240, New York, NY, USA, 2016. Association for Computing Machinery.

[98] Clemens Zeidler, Gerald Weber, Wolfgang Stuerzlinger, and Christof Lutteroth. Automatic generation of user interface layouts for alternative screen orientations. In *IFIP Conference on Human-Computer Interaction*, pages 13–35. Springer, 2017.

[99] Rares Vernica and Niranjan Damera Venkata. AERO: An extensible framework for adaptive web layout synthesis. In *Proceedings of the 2015 ACM Symposium on Document Engineering*, pages 187–190, 2015.

[100] Oussama Beroual, Francis Guérin, and Sylvain Hallé. Detecting responsive web design bugs with declarative specifications. In *International Conference on Web Engineering*, pages 3–18. Springer, 2020.

[101] Ibrahim Althomali, Gregory M Kapfhammer, and Phil McMinn. Automatic visual verification of layout failures in responsively designed web pages. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, pages 183–193. IEEE, 2019.

[102] Thomas A Walsh, Gregory M Kapfhammer, and Phil McMinn. ReDeCheck: An automatic layout failure checking tool for responsively designed web pages. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 360–363, 2017.